Abstract—We present a method to find an alternate path, after a link failure, from a source node to a destination node, before the Interior Gateway Protocol (IGP) has had a chance to reconverge in response to the failure. The target application is a small (up to tens of nodes) regional access subnetwork of a service provider’s network, which is a typical access scale encountered in practice. We illustrate the method and prove that it will find a path if one exists.

Index Terms—Routing protocols, alternate routing, network survivability.

I. INTRODUCTION

We present a method to find an alternate path, after a link failure, from a source node to a destination node. Since reconvergence of an Interior Gateway Protocol (IGP) can take hundreds of milliseconds, there is a need for a method that will find an alternate path in less time than this. The target application is a small (up to tens of nodes) access subnetwork of a service provider’s network, which is a typical scale encountered in practice; a service provider typically has many such small regional access networks.

Consider a source node $s$ sending data to destination node $d$. Suppose some link $(i,j)$ on the shortest path from $s$ to $d$ fails. An IGP will find an alternate path from $s$ to $d$ that avoids $(i,j)$ (assume such a path exists). However, IGP re-convergence may take hundreds of milliseconds or even seconds, and the packet loss during this time period may be unacceptable. Fast Re-Route (FRR) methods establish a new path from $s$ to $d$ in much less time than required for IGP re-convergence.

There are several available FRR methods. One method, used in conjunction with label-based forwarding (e.g., LDP), creates an RSVP primary tunnel between each pair of nodes. In addition, a bypass tunnel is pre-defined for each arc $(i,j)$; the bypass tunnel for $(i,j)$ is a path from $i$ to $j$ that is physically disjoint from the link $(i,j)$. When the packet reaches node $i$ and link $(i,j)$ has failed, a local repair forwards the traffic along the bypass tunnel for $(i,j)$; when the packet reaches node $j$, it continues on the path defined by the RSVP primary tunnel. The disadvantage of this approach is that, for a network of $N$ nodes and $A$ arcs, $N(N-1)$ uni-directional primary tunnels and $2A$ uni-directional bypass tunnels are required.

An alternative to building tunnels is to use a Loop Free Alternative (LFA) method ([2], [4]). For any two nodes $i$ and $j$, let $c^*(i,j)$ be the minimal distance between $i$ and $j$. Suppose node $n$ is a neighbor of $s$ (i.e., they are connected by a single arc). Then the neighbor $n$ of $s$ is an LFA for destination $d$ if

$$c^*(n, d) < c^*(n, s) + c^*(s, d).$$

That is, $n$ is an LFA if the shortest path from $n$ to $d$ does not return to $s$ on the arc $(n, s)$. To ascertain whether an LFA exists for a given $s$ and $d$ it suffices to determine if (1) holds for some neighbor $n$ of $s$.

A simple example where LFAs always exist is a three node network whose arc lengths satisfy the triangle inequality. A simple example where LFAs never exist is the square in Fig. 1 where all arcs have the same cost. If arc $(s, d)$ fails, then for the other neighbor $u$ of $s$, before the IGP re-converges the paths $u \rightarrow s \rightarrow d$ and $u \rightarrow v \rightarrow d$ have the same cost, so (1) fails to hold for source $s$ and destination $d$. There are several proposed enhancements of LFA, and several alternative methods, to extend the set of topologies for which FRR can be achieved; we now review some of them.

One way to enlarge the set of topologies for which LFA holds is to add a small number of tunnels, used only in the event of a failure. To illustrate this method, called remote LFA [3], consider Fig. 2 where all links on the ring have the same cost $c$. Suppose we are routing from $A$ to $B$, link $(A, B)$ fails, and the IGP has not re-converged. Then there is no LFA, since if the packet is delivered to $F$, this node sends the packet right back to $A$. The remote LFA approach creates an $(A, D)$ tunnel to be used only if $(A, B)$ fails; packets taking this tunnel and arriving at $D$ will be forwarded to $C$ and then to $B$. Hence with this extra tunnel, $D$ is an LFA for traffic from $A$ to $B$. Similarly, a $(D, A)$ tunnel makes $A$ an LFA for source $D$ and destination $C$ in case arc $(D, C)$ fails. Considering all possible source/destination pairs, a total of six tunnels, to be used only in case of a link failure, are required. In general, although the remote LFA approach uses fewer tunnels than the full mesh tunnel-based RSVP approach, it still requires a case by case analysis.

Fig. 1. No LFA for the square.

Fig. 2. Remote LFA.
analysis of the topology to determine where tunnels need to be added.

The problem of finding the minimum number of edges to add, in order to achieve full LFA coverage, is studied in [10], under the assumption that the edges in the original graph, and the edges to be added, have the same cost. They formulate this as an integer linear program with $N^3$ variables, and show that it is NP-complete. They propose a greedy heuristic with $O(N^3(N^2 - M^2))$ complexity, which in every iteration adds the edge that most increases LFA coverage.

Another extension of the basic LFA methodology is the u-turn method [1]. Consider again Fig. 2, with $A$ as the source, $C$ as the destination, and where link $(A, B)$ failed. Ignore the $(A, D)$ link in this figure; this link was added to illustrate remote LFA. Node $F$ is not an LFA. But $E$, a neighbor of $F$, has the option of forwarding packets to $D$, who will then forward them to the destination $C$. That is $F$ can break the “u-turn” by forwarding to $E$. In general, if $n_3$ is a neighbor of $s$, and if $n_2$ is a neighbor of $n_1$ such that $c^*(n_2, t) < c^*(n_2, s) + c^*(s, t)$, then $n_1$ should forward the packet to $n_2$. While this method is an improvement of LFA, it also is not guaranteed to find an alternate route, even if one exists.

The Recursive Loop-Free Alternative (RLFA) method [9] calculates alternate paths that are maximally edge disjoint from the shortest path. Let $C$ be the sum of all arc costs in the network, and let $\mathcal{P}(s, d)$ be the shortest path between $s$ and $d$. The method adds $C$ to each link on $\mathcal{P}(s, d)$ and then calculates a new shortest path for each pair of nodes. These local alternative next hops are used to route packets through a failure-free branch of the shortest path tree rooted at $s$. While this method is guaranteed to provide an alternate path (if one exists) for any single link failure, it does require a large number of shortest path computations to compute the local alternative next hops.

The MRC method [7] computes a set of backup configurations, such that each link or node is excluded from forwarding packets in exactly one configuration. Each configuration is generated by setting link costs appropriately, and computing shortest paths using these costs. Each router maintains a separate forwarding table for each configuration. When a link or node fails, the corresponding backup configuration is selected, the packet is marked with this configuration, and is forwarded based upon this marking. The complexity of this method is $O(n\Delta N A)$, where $n$ is the user specified number of bypass configurations desired, $\Delta$ is the maximum node degree, and $N$ and $A$ are the number of nodes and arcs, respectively.

Centralized alternate routing methods are employed in [8], which also reviews the FRR literature and presents theoretical results on the existence of alternate routes. Another class of related methods ([5],[11]) employs link-reversal, which also reviews the FRR literature and presents theoretical formulations.

To summarize, extensive experience at AT&T, both in the lab and in production, has shown that the time for an IGP to converge after a link failure can be hundreds of milliseconds. We want to establish an alternate path in less time than this. LFA, without tunnels, cannot handle such simple cases as the square topology of Fig. 1. Tunnel based methods require substantial overhead, in tunnel creation (often accomplished by scripting), in operations/maintenance to periodically check the health of each tunnel, and in the imposition of extra state on line cards to support forwarding packets along the tunnels. Moreover, in applications such as virtualization of VPNs, when the assignment of PE (provider edge) routing functionality is moved from one virtual machine to another, redesign of the tunnels is required, and the addition of a new node to the VPN requires building tunnels to the new node. A method not requiring tunnels is much simpler. Finally, the alternative of storing precomputed paths requires preprocessing and storing these alternate paths, and thus shares many of the disadvantages of tunnels.

II. THE METHOD

We now present the details of the method. Let $G = (N, A)$ be an undirected connected graph with node set $N$ and arc set $A$. For $x \in N$, let $N(x)$ be the set of neighbors of $x$, where a neighbor of $x$ is a node one arc away from $x$. We associate with each undirected arc $(i, j) \in A$ a cost $c(i, j)$, and require each $c(i, j)$ to be a positive integer. (The integer valued restriction can always be met by approximating, to the desired accuracy, each arc cost by an improper fraction, and then multiplying all the fractions by the least common multiple of the fraction denominators.) For $i, j \in N$, let $c^*(i, j)$ be the cost of the shortest path in $G$ between $i$ and $j$. When using Route$(s, d)$ for fast re-route in the event of an arc failure, which is the target application, $c^*(i, j)$ represents the shortest path cost before the IGP has reconverged in response to the link failure. Let $s$ be a given source node, and $d$ be a given destination node. In procedure Route$(s, d)$ below, $\mathcal{P}$ is an ordered list of nodes that have been visited, and $P \leftarrow \{\mathcal{P}, x\}$ means that $x$ is inserted after the rightmost element in $\mathcal{P}$. Also, $\Delta(n)$ is the multiplicity of node $n$, indicating how many times $n$ has been visited by the current packet.

```plaintext
procedure Route(s, d)
1  initialize: $\mathcal{P} = \emptyset$, $\Delta(n) = 0$ for $n \in N$, and $x = s$;
2  while ($x \neq d$) {
3    Let $\mathcal{Y} = \{y \in N(x) | \Delta(y) = \min_{n \in N(x)} \Delta(n)\}$;
4    Pick any $y \in \mathcal{Y}$ for which the sum
5      $c(x, y) + c^*(y, d)$ is smallest;
6    Set $\Delta(x) \leftarrow \Delta(x) + 1$, $\mathcal{P} \leftarrow \{\mathcal{P}, x\}$,
7      and send the packet and $\mathcal{P}$ from $x$ to $y$;
8    Set $x \leftarrow y$;
9  }
```

In words, if $x$ is the latest node to receive the packet, we find the set of neighbors of $x$ with lowest multiplicity. From this set, we pick the neighbor $y$ for which $c(x, y) + c^*(y, d)$ is smallest. We append $x$ to $\mathcal{P}$, augment the multiplicity of $x$ by 1, and send the packet and $\mathcal{P}$ to $y$. Note that $\mathcal{P}$ can be used to compute the multiplicities; e.g., if $\mathcal{P} = \{s, f, g, f, s, d, c, a, c, f, g, s, d, c, a\}$ then $\Delta(a) = 2$, $\Delta(c) = 3$, $\Delta(d) = 2$, $\Delta(f) = 3$, $\Delta(g) = 2$, and $\Delta(s) = 3$. This example also shows that instead of sending $\mathcal{P}$ to the next node, we could instead send only the nodes visited and their multiplicities, e.g., we could send $\{\Delta(a) = 2, \Delta(c) = 3, \Delta(d) = 2, \Delta(f) = 3, \Delta(g) = 2, \Delta(s) = 3\}$. Note that we
optionally could add a step, immediately following Step 2, which says that if \( d \in N(x) \) then forward the packet to \( d \).

Steps 3 and 4 are illustrated in Fig. 3. The neighbors of \( x \) are \( p, q, \) and \( r \); of these, \( p \) and \( q \) have the lowest multiplicity. Since \( c(x, q) + c^*(q, d) < c(x, p) + c^*(p, d) \), the packet is next forwarded to \( q \).

If we apply the method to the square of Fig. 1, with source \( s \), destination \( d \), and link \((s, d)\) failed, \( s \) will forward the packet to \( u \). Since now \( \Delta(v) = 0 \) and \( \Delta(s) = 1 \), then \( u \) forwards the packet to \( v \). Since now \( \Delta(d) = 0 \) and \( \Delta(u) = 1 \), then \( v \) forwards the packet to the destination \( d \). Thus the method easily computes an alternate route for the square, which is a case where LFA fails.

A more interesting example is provided by Fig. 4, where all arcs have cost 1, except for \((z, d)\) with cost 10. Suppose \((s, d)\) fails and the IGP has not yet re-converged. If in Step 4 of Route\((s, t)\) we break ties by picking the lexicographically smallest node (e.g., closer to “a” in the alphabet), then the path taken is \( s \rightarrow u \rightarrow v \rightarrow w \rightarrow x \rightarrow y \rightarrow z \rightarrow d \). If in Step 4 of Route\((s, t)\) we break ties by picking the lexicographically largest node (e.g., closer to “z” in the alphabet) then \( u \) forwards the packet to \( z \), and \( z \) forwards the packet to \( y \), since \( \Delta(y) = \Delta(d) = 0 \) but \( c(z, y) + c^*(y, d) = 1 + 4 < c(z, d) = 10 \). The packet will eventually reach \( d \), but by a longer path than with the “lexicographically smallest” rule.

To implement this method, the packet header can be expanded to specify the multiplicity of each node. For a small network (e.g., a regional access network, which was the application motivating this method) the memory required for this is small. As for throughput considerations, MPLS networks today routinely process packets with even 5 labels in a stack, without sacrificing throughput. For small access networks, the processing needed to determine the neighbor to whom the packet should be sent (Steps 3 and 4 above) can also be done without sacrificing throughput.

### III. Convergence Proof

In this section we prove that Route\((s, d)\) will find a path in \((N, A)\) from \( s \) to \( d \), if such a path exits. For \( x \in N \), let \( \deg(x) := |N(x)| \), that is, the number of neighbors of \( x \). Let \( N := |N| \) be the number of nodes in the network. Note that the convergence proof makes no use of the arc costs, and thus is independent of the rule used in Step 4 of Route\((s, d)\) to choose among those neighbors of \( x \) with lowest multiplicity.

**Lemma.** For \((x, y) \in A\), and at any point during the packet’s path, \( \Delta(y) \geq \frac{\Delta(x)}{\deg(x)} - 1 \).

**Proof.** Each time the packet leaves \( x \) to some neighbor \( y \) of \( x \), on the next step either \( \min_{n \in N(x)} \Delta(n) \) will increase, or the cardinality of the set \( \{ y \in N(x) | \Delta(y) = \min_{n \in N(x)} \Delta(n) \} \) will decrease. The second alternative can happen at most \( \deg(x) \) times before the cardinality of this set drops to the value 1, so the first alternative must happen at least once every \( \deg(x) \) times.

The packet must have had a next step after leaving \( x \) at least \( \Delta(x) - 1 \) times. Thus, \( \min_{n \in N(x)} \Delta(n) \) must have been increased at least \( \left\lfloor \frac{\Delta(x) - 1}{\deg(x)} \right\rfloor \) times. But \( \Delta(y) \) is included in that min, so \( \Delta(y) \geq \left\lfloor \frac{\Delta(x) - 1}{\deg(x)} \right\rfloor = \frac{\Delta(x)}{\deg(x)} - 1 \).

**Corollary 1.** For \((x, y) \in A\), and at any point during the packet’s path, if \( \Delta(x) \geq 2 \deg(x) \), then \( \Delta(y) \geq \frac{\Delta(x)}{\deg(x)} \).

**Proof.** By the Lemma, \( \Delta(y) \geq \frac{\Delta(x)}{\deg(x)} - 1 = \frac{\Delta(x) - \deg(x)}{\deg(x)} \). If \( \Delta(x) \geq 2 \deg(x) \), then \( \frac{\Delta(x) - \deg(x)}{\deg(x)} \geq \frac{\Delta(x)}{2 \deg(x)} \).

**Corollary 2.** If \( x = n_1, n_2, n_3, \ldots, n_k = y \) is a path from \( x \) to \( y \), and if \( \Delta(x) \geq \prod_{i=1}^{k} (2 \deg(n_i)) \), then \( \Delta(y) \geq \Delta(x) / \prod_{i=1}^{k} (2 \deg(n_i)) \).

**Theorem 1.** If there is a path from \( s \) to \( d \), then after a finite number of steps, the packet will reach \( d \).

**Proof.** Let \( T = \prod_{n \in N} (2 \deg(n)) \). Since the multiplicity of some node increases by 1 after each step, then after \( N \times T \) steps, some node \( z \) must have \( \Delta(z) \geq T \). Since the graph is symmetric, and since there is a path from \( s \) to \( z \) and from \( s \) to \( d \), there must be a path from \( z \) to \( d \). Let this path be \( z = n_1, n_2, n_3, \ldots, n_k = d \). But then \( \Delta(z) \geq T \geq \prod_{i=1}^{k} (2 \deg(n_i)) \), so by Corollary 2, \( \Delta(d) \geq T / \prod_{i=1}^{k} (2 \deg(n_i)) \geq 1 \). But if \( \Delta(d) \geq 1 \), the packet must have reached \( d \).

Consider Fig. 5, where all arc costs are 1, and suppose link \((b, d)\) fails. Without a stopping criterion, Route\((s, d)\) will create the cycle \( s \rightarrow a \rightarrow b \rightarrow a \rightarrow s \) and repeat this cycle indefinitely. However, by examining path \( P \) to detect cycles, we can provide a stopping criterion. Let \( C \) be a finite length sub-string of \( P \) such that the first and last elements of \( C \) are both the source node \( s \) (so \( C \) is a cycle). For example, for Fig. 5 we would choose \( C = \{s, a, b, a, s\} \). Let \( C^0 \) be the substring of \( C \) obtained by deleting the initial element \( s \), e.g., \( C^0 = \{a, b, a, s\} \) for this example. Let \( \{C, C^0\} \) be the string obtained by appending \( C^0 \) to the end of \( C \); e.g., for our example, \( \{C, C^0\} = \{s, a, b, a, s, a, b, a, s\} \). The following theorem says that if at some point \( P \) consists of two identical
cycles, then there is no path from \( s \) to \( d \) and \( Route(s, d) \) can be stopped.

**Theorem 2.** If for some finite length string \( C \), whose first and last elements are \( s \), procedure \( Route(s, t) \) generates a path \( \mathcal{P} \) such that \( \mathcal{P} = \{C, C^0\} \), then there is no path from \( s \) to \( d \) and \( Route(s, d) \) can be terminated.

**Proof.** Suppose such a string \( C \) exists. Suppose \( C \) contains a total of \( k + 1 \) occurrences of node \( s \). Since \( C \) begins and ends with \( s \), then \( k \geq 1 \). For \( 1 \leq i \leq k \), let \( C_i \) be the substring of \( C \) starting with the \( i \)-th occurrence of \( s \) and ending with the \((i+1)\)-st occurrence of \( s \). For example, if \( C = \{s, x, y, z, x, s, e, h, f, h, g, s\} \) then \( C_1 = \{s, x, y, z, x, s\} \) and \( C_2 = \{s, e, h, f, h, g, s\} \). Each of the \( k \) substrings \( C_i \) defines a loop that either (case 1) leaves \( s \) and returns to \( s \) from the same “loop gateway node” (e.g., \( C_1 \) leaves and returns to \( s \) from \( x \)), or (case 2) leaves \( s \) and returns to \( s \) from different “loop gateway nodes” (e.g., \( C_2 \) leaves \( s \) from \( e \) and returns to \( s \) from \( g \)). These two possibilities are illustrated in Fig. 6, where case 1 is illustrated by the loop leaving/returning at the loop gateway node \( n_1 \), and also by the loop leaving/returning at loop gateway node \( n_4 \), and case 2 is illustrated by the loop leaving at the loop gateway node \( n_2 \) and returning from the loop gateway node \( n_3 \).

Let \( \mathcal{N}_P \) be the set of nodes visited in \( \mathcal{P} \). For example, if \( \mathcal{P} = \{s, x, y, z, x, s, e, h, f, h, g, s\} \) then \( \mathcal{N}_P = \{s, x, y, z, e, h, f, g\} \). Since each of the \( k \) loops in \( C \) is traversed twice, the multiplicity of each node in \( \mathcal{N}_P \) must be at least two. Suppose, contrary to the statement of Theorem 2, that there is a path \( \mathcal{P}_{s, d} \) from \( s \) to \( d \). We can assume without loss of generality that \( \mathcal{P}_{s, d} \) contains no loops.

Let \( x \) be the successor node to \( s \) on path \( \mathcal{P}_{s, d} \). We consider two cases. First, suppose \( x \notin \mathcal{N}_P \). Consider the iteration where \( Route(s, d) \) has just generated the partial path \( C \) and returned to \( s \). Then each node in \( \mathcal{N}_P \) has multiplicity at least 1, but \( x \) has multiplicity 0, since \( x \notin \mathcal{N}_P \). By Step 3 of \( Route(s, d) \), node \( x \) will be picked before again selecting any node in \( \mathcal{N}_P \). But \( Route(s, d) \) did not pick \( x \), and instead ultimately generated the path \( \{C, C^0\} \). Hence it must be that \( x \in \mathcal{N}_P \).

So consider the second case, where we assume \( x \in \mathcal{N}_P \). Suppose \( x \) lies on loop \( i \), where \( 1 \leq i \leq k \). No node on loop \( i \) is the destination \( d \), for otherwise the algorithm would have halted. So there must be a last node \( y \) which lies on \( \mathcal{P}_{s, d} \) and which also lies on loop \( i \). (It might be that \( y = x \).) Let \( z \) be the successor node to \( y \) on the path \( \mathcal{P}_{s, d} \), so \( z \) is not on loop \( i \). (See Fig. 7.)

We consider two sub-cases. Suppose first that \( z \notin \mathcal{N}_P \). Consider the iteration where the path \( \mathcal{P} \) generated by \( Route(s, d) \) has traversed loop \( i \) once, and then later returns to \( y \). Then each node on this loop has multiplicity at least 1, but \( z \) has multiplicity 0, since \( z \notin \mathcal{N}_P \). By Step 3 of \( Route(s, d) \), \( z \) will be picked before again selecting a node on loop \( i \). But \( Route(s, d) \) did not pick \( z \), and instead traversed this loop a second time. Hence it must be that \( z \in \mathcal{N}_P \).

Finally, consider the second sub-case, where we assume \( z \in \mathcal{N}_P \). Since by definition \( z \) is not on loop \( i \), then \( z \) must lie on some other loop, say loop \( j \), where \( j \neq i \). By the same arguments as above, the path \( \mathcal{P}_{s, d} \) must leave loop \( j \), and when it does so it must immediately visit another loop. This jumping between loops can occur at most \( H \) times, where \( H \) is the number of arcs in path \( \mathcal{P}_{s, d} \). Thus \( \mathcal{P}_{s, d} \) never reaches \( d \), which contradicts the definition of \( \mathcal{P}_{s, d} \). Hence this second sub-case \( z \in \mathcal{N}_P \) yields a contradiction, and hence the second case \( x \in \mathcal{N}_P \) cannot hold.

Having shown that both \( x \notin \mathcal{N}_P \) and \( x \in \mathcal{N}_P \) cannot hold, we conclude that there is no path \( \mathcal{P}_{s, d} \) from \( s \) to \( d \).

**ACKNOWLEDGMENT**

The convergence proof for Theorem 1 was provided by David Applegate. The authors thank the reviewers for their valuable comments and suggestions.

**REFERENCES**