Improving Service Availability during Link Failure Transients through Alternate Routing

Siridhar Vellanki, A.L. Narasimha Reddy

Abstract

A link failure in a network results in interruption of service for many end users. Currently, routers recompute their routing tables to find new routes around the failure. However, the network drops packets in the transient period starting at the time at which an attached link fails to the time until the routing tables are recomputed. This effect is significant in the high bandwidth backbone networks and more critical when voice calls are supported on IP networks. Our research aims to reduce the packet losses during this transient period. We propose a simple alternate/deflection routing scheme activated by the routers at the time of link failure until new routes are computed. The proposed scheme will make the networks more robust and help in improving the service availability.

Index Terms: Alternate/Deflection routing, Link failure, Backbone networks, Sink tree.

1 Introduction

Many real-time applications like VoIP require uninterrupted service from the network. In [1] the authors have investigated the issues involved in the deployment of a voice service over a tier-1 IP backbone network. It was observed that the level of congestion in the backbone is negligible and did not have impact on the voice quality. It was also observed that voice packets do not need any specific handling but require new mechanisms to provide protection against link failures. Though link failures do not occur frequently, their effects are significant in backbone networks.

A link can be marked as down by a router due to a fiber cut or inoperable interfaces at either end of the link or due to hardware problems in the router. A link may also be down due to false failures when link state packets fail to reach the routers at either end of the link. In such a case, when the router gets the next link state packet, the link is marked back as up and hence, results in link failure for a duration of link state update period.

Currently, routers recompute their routing tables to find new routes around the failure. However, routers drop packets in the transient period until the routing tables are recomputed to route packets around the failure. This transient period, referred as routing convergence delay or service disruption time, is the duration between a link failure and the instant when router resumes forwarding the packets headed on the failed link. It has been observed in [2, 3, 4] that typically this transient period is around 6 to 7 seconds at the backbone routers. It has also been observed that significant fraction (50%) of the failures last less than a minute [2]. For these failures, the transient periods are a significant part of the period of the link failure.

The increasing usage of Voice-over-IP (VoIP) applications and the stringent requirement on the backbone networks to meet the Service Level Agreements requires better service availability in the network. Particularly, we need mechanisms to prevent or reduce the packet drops by the routers during the transient periods of link failures. Instead of dropping the packets during the transient period, routers could forward them to other links so that they can reach the destination through a different path. We term this action as deflection of the packet. Since most of the links in the backbone are utilized less than 50% [2], by deflecting the packets on other links we could improve service availability. However, care should be taken so that packets would not circulate in loops due to this deflection. Packets cannot be deflected randomly onto other links because that could result in an out-of-sequence delivery at the destination if packets of the same flow are deflected to different links. Also changes to the basic routing protocols and packet forwarding mechanisms should be minimized with such wide spread deployment of present routing protocols.

In this paper, we present a simple deflection routing scheme where each node computes a map for each of its links, to an alternate link. This alternate link will be used to forward a packet when the original link fails. Since the number of links at a router is typically much smaller than the number of destinations in a routing table, the alternative link table is much smaller than a typical routing table. Such an approach leaves the existing routing tables unmodified. A packet is deflected if a node detects a link failure or if the incoming interface and the outgoing interface of the packet are the same. Alternate link table is computed at the time when routing tables are computed. We use the terms alternate routing and deflection routing interchangeably in this paper, when the packets take a different path from the normal shortest path, as a result of a link failure.

The basic idea is illustrated through an example network in Fig. 1. Normal routing tables and alternate link tables for all the 5 nodes in the network are shown in the figure. Each node (example 'A') is connected to a number of hosts (a1, a2, a3 and a4 etc.). Consider an example where node 'D' has to send a packet to node 'B' and the link 9 is down. When node 'D' finds that the link to the next-hop according to its routing table for node 'B' is down, it checks its alternate link table. The alternate link table suggests that link 8 should be used as an alternate link when link 9 cannot be used to forward a packet. Hence 'D' forwards the packet on link 8. Node 'C', after receiving the packet, forwards it on link 7, in the normal way, according to its routing table. Therefore, in this example, the packet is deflected to an alternate link when the original link fails and after one deflection, the packet is back on a shortest path to the destination. The alternate path has required two additional hops for the packet to reach the destination. D's routing table indicates that link 9 is used for forwarding packets to destinations 'A' & 'B' and hence, link 8 will be used as an alternate for both these destinations.

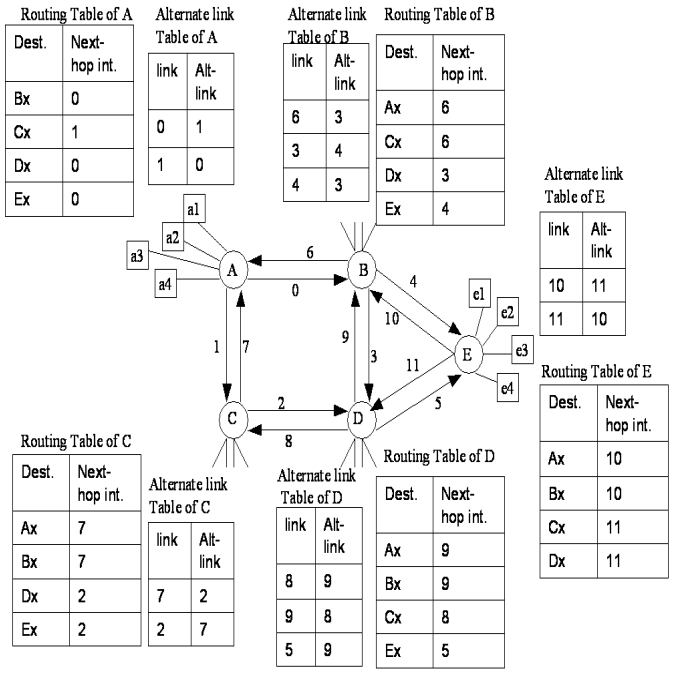

Figure 1: Sample network with routing tables and alternate link tables

If alternate paths are computed after a failure has become evident, packets will have to be dropped until the alternate link maps or reconstructed routing tables become available. The effectiveness of an alternate routing mechanism is determined by the speed at which a detour path is established for forwarding packets after a failure. Hence, the alternate link map should be pre-computed so that it can be used immediately after a failure is detected. We need a scheme where a router does not have to communicate with other neighboring routers or does not have to do any computation, before it can start deflecting the packets. Our aim is to make the service disruption time zero in order to continue providing service during the transient periods of link failures. Our aim is also to make these deflection paths to be as optimal as possible thus minimizing the cost in forwarding packets through alternate paths. Our approach does not require routers to encapsulate packets or to modify the packet headers in anyway to deflect packets. Our approach ensures that deflection does not cause out-of-order delivery problems.

This paper is structured as follows. In Section 2, we present our algorithm. Evaluation and results are presented in Section 3. Alternate routing in hierarchical networks and previous work on alternate routing are discussed in Section 4. Section 5 presents conclusions and discusses future work.

2 Algorithm

In order to deflect packets quickly around a failed link, our algorithm relies on the routers to make decisions locally, without requiring the state of the failed link to be propagated through out the network. To keep the alternate link table small, we deflect all the packets headed on a link onto another link irrespective of the destination address.

Our work is primarily motivated by the following goals: (a) to employ normal forwarding mechanisms i.e., no source routing, even when packets are being deflected, (b) to utilize existing routing tables as much as possible for routing packets around the failure and, (c) to require no information exchange among routers before packets can be deflected (routed) around the failure. Our deflection routing algorithm is designed based on these guidelines. We assume that the knowledge of a link failure is limited to the routers directly connected to the failed link. As a result, when a packet arrives at any router (even after deflection) in the network, the router continues to consult its existing routing tables to forward the packet. A link failure may only become visible to another router (not directly connected to the failed link) when the outgoing link happens to be the same as the incoming link of the deflected packet. This leads the router to conclude that a link failure elsewhere is being "reflected" on one of its links and that the router has to route around the failure.

The alternative link tables are computed whenever routing tables are computed and they are consulted when a link failure is detected at a router. A link failure could be detected by an immediate router through hardware status or by monitoring the status of the queue at the output interfaces. The alternative link tables are used until the new routing tables are computed (after the link state of the failed link has been propagated through the network). Once the new routing tables are computed, forwarding along the alternative link tables is discontinued.

Routing table entries are easily understood through sink trees. We illustrate our approach to deflection through a number of examples on sink trees. Later, we present the algorithm that computes the alternate link maps.

2.1 Illustration by Examples

Basic idea of our deflection routing scheme will be explained using sink trees. Sink tree of a node 'R' in a network is a tree that has only the links, which are used by all other nodes to reach 'R' or vice versa in that network. The paths between the nodes in the sink tree are the shortest paths between them. We assume that all the links have unit weights in the following discussion.

Fig. 2 shows a sink tree of a node 'R' in a network. The edges shown in dotted lines termed as Bridges are links in the network, which are not part of node R's sink tree. Bridges are links that allow

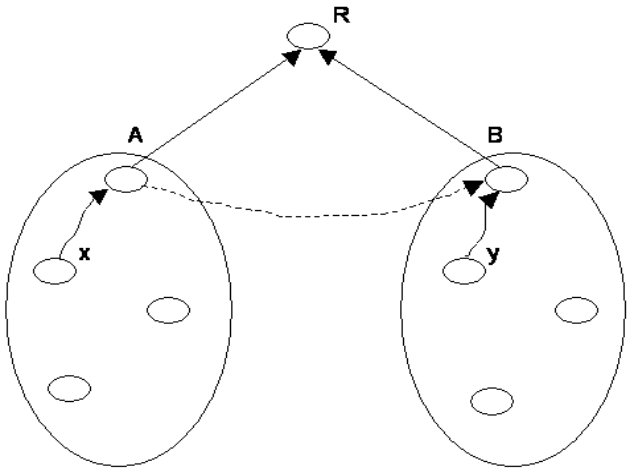

Figure 2: Sink tree of node 'R' in a network (Illustration 1)

packets to be deflected from one sub-tree of the sink tree to another sub-tree. All the nodes in the cloud under 'A', will normally use the link (A, R) to reach 'R' and all the nodes in the cloud under 'B', will normally use the link (B, R) to reach 'R'.

Consider that we need to find an alternate link for link from 'A' to 'R' (A, R) due to its failure. To deflect the packet at 'A' so that it can reach the destination, the packet has to be sent through a bridge to the other side of the sink tree as soon as possible. The packet can then reach the destination through a shortest path using the existing routing tables. For the example shown in the Fig. 2, we deflect the packet through link (A, B), which is a bridge. Once the packet reaches node B, node B looks up its normal routing table and forwards the packet on the link (B, R). Therefore, the alternate link map needed for link (A, R) is (A, B), which means the packets headed on link (A, R) are deflected on to link (A, B), when link (A, R) fails and the routing tables are not yet recomputed.

Even if the packet is destined to a node further away from 'R', node 'R' being just an intermediate node, deflection of the packet onto link (A, B) will ensure that packet reaches the destination without loops. This is explained with an example in Fig. 3. A packet would normally take the link (A, R) going from the source to the destination with the existing routing tables. When link (A, R) fails, it is deflected to B. The length of the shortest path from 'B' to the destination is less than or equal to (l2 + 1). If B's routing table pointed to any node along the path from (source to node 'A'), then its path length would be greater than (l2 + 1). This cannot be the case if the routing tables are computed correctly since a shorter path to the destination with length (l2 + 1) exists through 'R'. In other words, whenever a packet has to cross link (A, R) for any destination, and if the link (A, R) is down, the same derouting path can be taken.

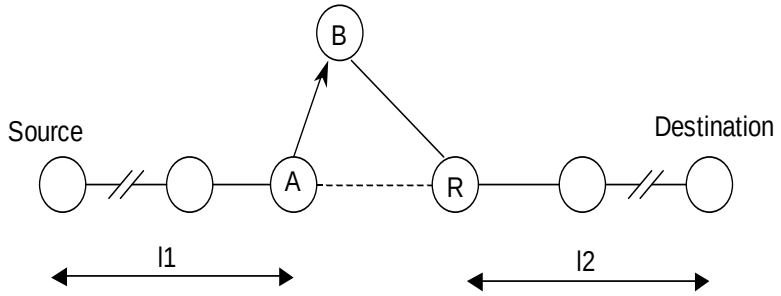

Figure 3: Example showing the generality of bridges

A deflected packet will not come back to previously visited nodes unless it is purposefully deflected. An example of such a scenario is illustrated through Fig. 4. Consider a packet heading on path B->A->R reached node 'A' and link (A, R) is down. Node 'A' has no other option but to deflect the packet back to node 'B'. After node 'B' gets the deflected packet, it will know that something is wrong down the line because the incoming and the outgoing interfaces of this packet are the same. Here, failure of link (A, R) is reflected onto link (B, A). Hence it looks up alternate link of (B, A) and deflects the packet to node 'C'. Node 'C' can then forward the packet to node 'R'. Hence, to avoid routing loops each node utilizes alternate link tables when the incoming interface is same as the outgoing interface based on the current routing tables. As a result, a packet might have to trace back, part of the path traversed if no other option is present at the point of a failure.

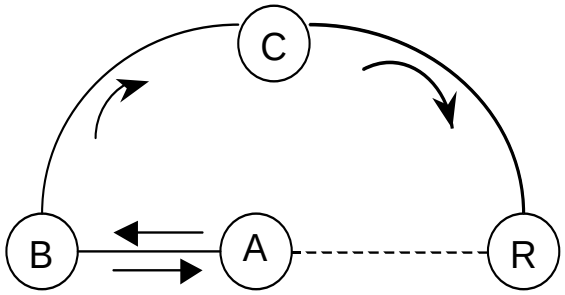

Figure 4: Sample network illustrating loop-back case

Fig. 5 shows another example of a sink tree of node 'R' in a network. Consider again, we need to find an alternate link for link (A, R) and there is a packet headed on that link. In this example, there is no bridge from node 'A' to node 'B', or to any other node in the cloud under 'B'. Deflecting the packet from node 'A' to any of the nodes in its own cloud is not sufficient because the packet would come back to 'A' in such a case.

For example, if the packet is deflected onto link (A, x), node 'x' will have node 'A' as a next-hop for

'R' in its routing table causing the packet to come back to 'A'. This results in a loop. In other words, for node 'x', its as if link (x, A) is down, since it cannot forward the packet on it. We term this as reflected failure. The failure of link (A, R) is reflected on to link (x, A). Therefore, the packet has to be deflected again at node x. This decision can be made by the fact that if the packet is forwarded on the link (x, A), the outgoing and the incoming interface of the packet will be the same, which can only happen on link failures or routing loops. This gives an indication of a failure and hence, the packet needs to be deflected again. If we have a bridge from node x to node B or any of the nodes in the cloud under B (say node y), the packet can be deflected on that link. From there, the packet will follow the shortest path according to the normal routing tables. So for this example, two alternate link mappings link (A, R) to (A, x) and link (x, A) to (x, y) are needed. Therefore a packet may have to be deflected multiple times, before it gets deflected onto a bridge, from where it can get forwarded based on normal routing tables. Whenever the packet has to cross link (A, R) in any other sink tree and if the link (A, R) is down, the same de-routing path is taken.

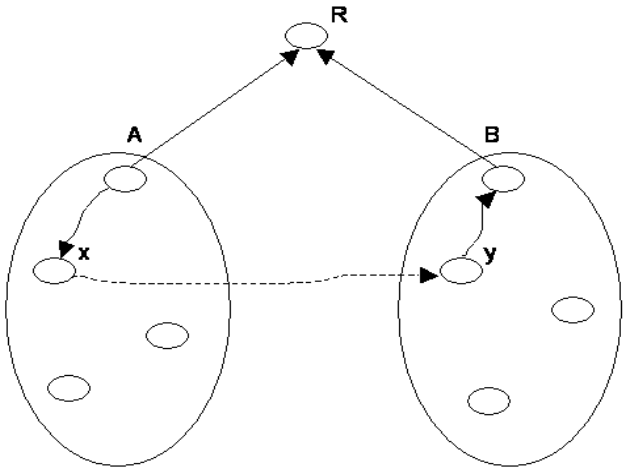

Figure 5: Sink tree of node 'R' in a network (Illustration 2)

An important point to note is that once the packet is deflected from link (A, R) to link (A, x), the packet need not necessarily be deflected again onto link (x, y). It depends on the destination of the packet. If the next-hop for the packet arriving from 'A' at node 'x' is pointing back to 'A', only then the packet is deflected on to link (x, y). For example, consider a node 'Z' directly connected to nodes 'x' & 'R' in Fig. 5. Consider node 'A' has a packet destined to node 'Z' and the link (A, R) is the next-hop link from 'A'. When link (A, R) is down, node 'A' deflects the packet to node 'x'. Since node 'x' is directly connected to 'Z', the outgoing interface for destination 'Z' is not pointing to link (x, A). Hence, it forwards the packet directly to 'Z', rather than deflecting it onto link (x, y).

But deflection of link (x, A) to (x, y) should also work when the link (x, A) fails, without creating loops. This checking has to be done while computing the alternate link for (x, A) in the sink tree of 'A'. In other words, all the alternate link maps computed due to reflected failures, have to be able to deflect packets around an actual failure of that link.

An example is presented below to illustrate the above scenario. Consider the network shown in Fig. 6. Let link (1, 3) be mapped as alternate link for link (1, 2) while computing alternate link for link (1, 2). Let links (0, 2) and (2, 1) be mapped as alternate links for links (0, 5) and (2, 0) respectively while computing the alternate link table for link (0, 5). Note that two maps have been set up while computing the alternate link table for link (0, 5), because of reflected failure.

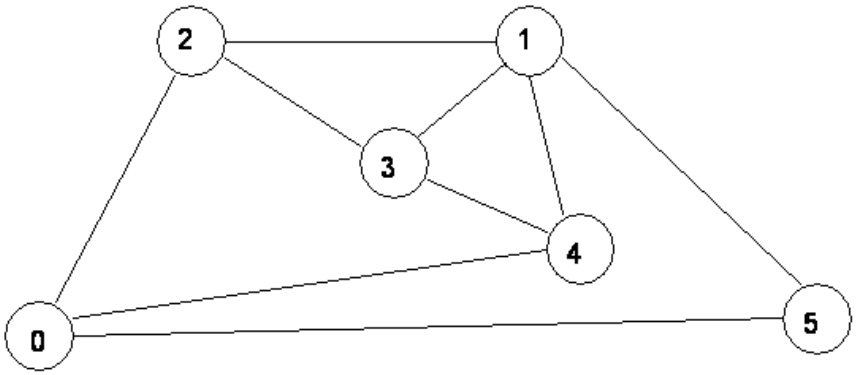

Figure 6: Sample network illustrating reflected failures

Now consider node '2' has a packet which has to be sent to node '0' and link (2, 0) is down. Since the link to next hop is down, node '2' looks up its alternate link table and forwards it to node '1'. Let node 1's next-hop for node '0' be node '2'. Since the outgoing interface is same as the incoming interface, node 1 looks up alternate link for link (1, 2) and forwards it to node '3'. Node 3's next-hop for node '0' is node '2' and because the outgoing interface is different from the incoming interface, node '3' forwards the packet to node '2'. Thus the packet has traveled in a loop from node 2->1->3->2. Therefore, we need to verify the alternate link maps created due to reflected failures, with its real failure. If a loop is found, the earlier alternate link map is removed and other options for deflection should be tried.

2.2 *FindAL* Algorithm

The main algorithm proceeds by finding an alternate link for all the links directly connected to the root of the sink tree of each node in the network. Each node has the entire knowledge of the network topology in link-state routing protocols and each node will run this algorithm and computes its alternate link table. We first explain the "Find Alternate Link" (*FindAL*) algorithm, which finds an alternate link for

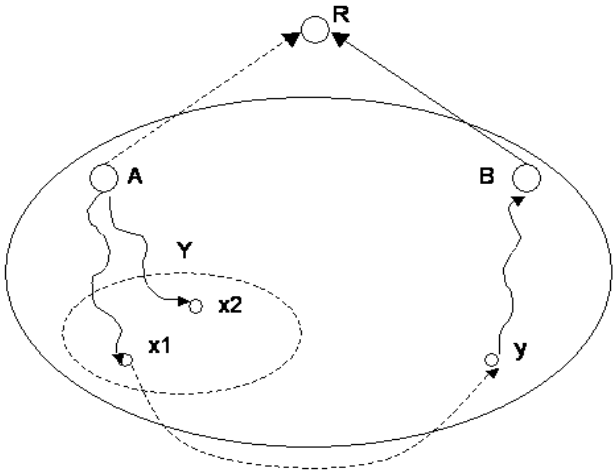

Figure 7: Sample network for illustration of algorithm *FindAL*

Step 1: Insert 'A' into list 'L';
mark node 'A';
$\pi(A) = R$;
Step 2: Pick a node, 'C' from the beginning of the list 'L';
If list 'L' is empty, then
return(unable to find alternate link);
Step 3: If link [C, π(C)] already has an alternate link map to link [C, x], then
mark node 'x';
$\pi(x) = C$;
insert 'x' in list 'L';
Else
Check for all nodes, 'Y' directly connected to 'C' (except the parent node)
If the next-hop node from node 'Y' to node 'R' is not marked, then
$\pi(Y) = C$;
go to Step 4;
Else
If node 'Y' is not marked, then
mark node 'Y';
$\pi(Y) = C$;
insert 'Y' into list 'L';
go to Step2;
Step 4: Back-trace using π's and establish the alternate link map.

Figure 8: *FindAL* Algorithm

a given link. This algorithm is called multiple times in the main algorithm to compute the alternate link maps for all the links. The main algorithm is explained in the next section.

Consider computing an alternate link for link (A, R) in a network shown in Fig. 7. Fig. 8 presents the pseudo code of the algorithm *FindAL*. A mechanism similar to Breadth First Search mechanism is employed until we find a bridge, which can be used as alternative path for link (A, R). Search starts at node 'A' looking at all the nodes (say 'Y') connected to it. By marking the visited nodes, we make sure that we do not look at those nodes again. This will help in avoiding routing loops while setting up the alternate link maps. If next-hop node in Y's routing table to node 'R' is not marked, then we have successfully found a bridge onto which packets can be deflected. If it is marked, then node 'Y' is inserted into the search list if it is not marked and the search continues. The algorithm uses existing alternate link map if possible in its search. This process is continued until we find a deflection path onto a bridge to other side of the sink tree where the packet can be forwarded along the shortest path. The algorithm sets up the alternate link maps by back-tracing the path taken after finding the bridge. It returns a failure if it cannot find a proper deflection path.

2.3 Flow of the Main Algorithm

Fig. 9 presents the pseudo code of the main algorithm. Initially all the nodes are arranged in a list. For each node (say 'A') from the list, alternate links are computed for all the links directly connected to it in its sink tree. Each link in the network will be mapped to its alternate link when all the nodes in the list are processed, i.e. when the list becomes empty. While processing each link of node 'A', it is first checked if an alternate link map already exists for that link. A map could have already been created, because of a reflected failure. In such a case, it verifies the correctness of that alternate link map. If the link is not yet mapped to an alternate link, then an alternate link is found by algorithm *FindAL*, explained earlier. If algorithm *FindAL* finds an alternate link, we move on to find a map for other links of 'A' or start processing the next node, if alternate links for all of A's links have been computed.

If the link already has a map to its alternate link, and if that alternate link can serve as a bridge for this link, the algorithm continues to find alternate links for other links. When an already existing alternate link does not work i.e., if it is not deflected to a proper bridge, its either because a loop is present or the current number of deflections is not sufficient for that link. This scenario was explained in Fig. 6 earlier. Hence, if there is no loop, it continues to find the rest of the deflection path using algorithm *FindAL*. If a loop exists or if algorithm *FindAL* cannot find an alternate link, the alternate link mappings causing the problem are removed and the nodes connected to those links are re-inserted at the end of the list 'L'. Node 'A' is also re-inserted at the beginning of the list and the algorithm

starts again with node 'A'. Essentially the order of the nodes in the list 'L' is changed and tried again.

One particular heuristic used by us is to order the nodes in the increasing order of degree of connectivity, when the algorithm begins to find the alternate links. Since nodes having smaller number of links have less choices for deflections, finding alternate links for those links first will likely result in finding more overall optimal deflection paths.

```
Step1: Create a list 'L' of all nodes, arranged in the increasing order of their degree of connectivity;
Step2: Pick a node, 'A' from the beginning of the list 'L';
        If list 'L' is empty, then
                return; // End of algorithm
Step3: For each link, 'l', node 'A' is connected to
                If link 'l' does not have alternate link, then
                        Find an alternate link using FindAL algorithm;
                        If FindAL algorithm could not find an alternate link, then
                                go to Step4;
                Else
                        Follow and check if the alternate link mapping is correct;
                        If the present mapping is not correct, then
                                Check if there is loop with the present alternate link mapping
                                If loop exists, then
                                        go to Step4;
                                Else // i.e., more deflection are needed
                                        continue and find alternate link using FindAL algorithm;
                                        If FindAL could not find an alternate link, then
                                                go to Step4;
        go to Step2;
Step4: Unlink the alternate link maps which is causing the problem;
        Insert node 'A' at the beginning of the list 'L';
        Insert the node whose alternate links are unmapped at the end of the list 'L;
        go to Step2;
```

Figure 9: Main flow of the algorithm

2.4 Complexity and Correctness

Current implementation of our algorithm limits the number of times search is re-done, i.e., we limit the number of times nodes are re-entered in the list, when the alternate link of their connected links are unmapped. This ensures that algorithm does not keep trying for alternate links when the connectivity

of the network is lost by a link failure. Our algorithm is considered correct if (a) it computes alternate link tables correctly or (b) when it reports a failure due to limited amount of search time. The efficiency of the algorithm is determined by the number of times it successfully computes the alternate link tables.

Computation of alternate link at each node, when search is limited as mentioned above, can be done in $O(E^2)$ time, where E is the number of edges in the network. Computation of a sink tree for a node can be done in $O(V lg E)$ where V is the number of nodes in the network. Hence, computation of sink trees for all the nodes can be done in $O(V^2 lg E)$ time. In practical implementation, the running time can be speeded up, by doing an incremental calculation of sink trees, i.e, computing sink tree of a node with the help of a sink tree of its neighboring node. Our alternate routing algorithm could be run separately on different levels of routing hierarchy to reduce the running time.

3 Performance Evaluation

3.1 Topology Generator

In order to test the performance of our alternate routing algorithm, we tested it on different network topologies. Our algorithm was successful in finding alternate links for the network topologies generated by the topology generator 97% of the time. We then, compared the performance in terms of (a) average number of hops and (b) worst case path length difference, with an ideal case where recomputed routing tables are available instantaneously to the router, after a link failure. We have used Brite [5, 6] topology generator [7] to generate different network topologies and tested our algorithm on those topologies. We used the Router Waxman model [8] in the topology generator to generate flat router-level topologies. The tests measure the effectiveness of our algorithm at one level of the routing hierarchy. We evaluate our alternate routing scheme by considering the failure of each link in a given network topology.

3.2 Evaluation

In our algorithm, the alternate link tables help the network in providing service without disruption when any link in the network fails. We evaluate our approach by considering the failure of each link for a given topology. We consider both the worst-case impact on path lengths and the impact on average path length, when all the source, destination pairs are considered. We evaluate the efficiency of our algorithm by comparing the path lengths with that of the case when routing tables are recomputed around the failure.

3.2.1 Average Number of Hops

In this section, we compare the average number of hops taken over paths from each (source, destination) pair in the network. We find the average value for the following three cases, (a) by following normal

routing tables without any link failures, (b) by following the alternate link table and deflection routing for each single link failure, and then (c) by following the recomputed routing tables, again for each single link failure in the network. We assume that a single link failure does not leave the network disconnected. In each network, we consider the failure of each link. For every link failure, we compute the average path length for each (source, destination) pair for the above mentioned cases (b) and (c). The average path length over all link failures is used as a performance measure. Worst case path length results are reported later in the section.

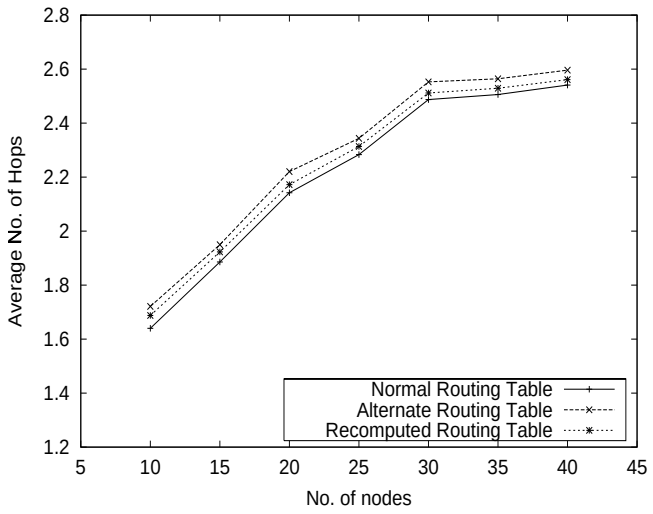

Figure 10: Average number of hops vs Number of nodes

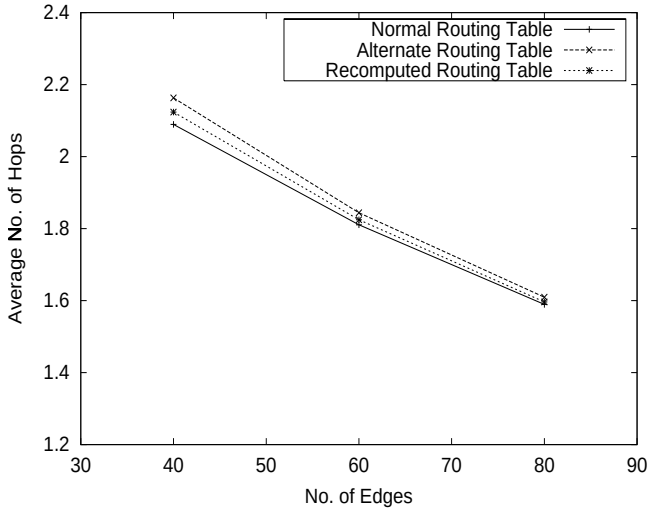

Figure 11: Average number of hops vs Number of edges

Average number of hops for all the three cases are plotted in the graph shown in Fig. 10, for network topologies with increasing number of nodes and constant degree of connectivity. In each network, the number of links is set to twice the number of nodes. The average number of hops for alternate link table case (b) is slightly higher than that of recomputed routing table case (c), which is in turn higher compared to the normal routing table case (a). It can be observed that the average path length difference with alternate routing and the recomputed routing tables is not significant and does not increase with increase in the number of nodes in the network.

Fig. 11 shows the average number of hops for network topologies with constant number of nodes and increasing degree of connectivity. It can be observed that the average path length difference with alternate routing and recomputed routing tables decreases with increase in degree of connectivity. When the degree of connectivity increases, there would be more choices to deflect the packets and hence, we will find a bridge soon without taking many deflections. We have run our algorithm against many topologies with different number of nodes and links. Due to space constraints, we present results from only a few cases.

Table 1: Average number of hops for different topologies with the same number of nodes and edges

Nodes	Edges	Average # of Hops		
		Normal routing table	Alternate link table	Recomputed routing table
14	28	1.85714	1.93407	1.8956
14	28	1.81319	1.89443	1.85989
14	28	1.9011	1.99038	1.93838
14	28	1.93407	2.01001	1.97449
14	28	1.86813	1.94506	1.90581
14	28	1.91209	1.98862	1.94427
14	28	1.85714	1.94348	1.90188
14	28	1.82418	1.89678	1.86224
14	28	1.87912	1.97174	1.92033
14	28	1.89011	1.96134	1.93053

Table 1 lists the values of average number of hops, for different network topologies. Topologies are generated by BRITE topology generator with 14 nodes and 28 edges. It is observed that the algorithm performs well across the different topologies. Table 2 lists the average number of hops for network topologies by increasing the number of nodes from 10 to 20 with constant degree of connectivity. The values of average number of hops with alternate routing are consistently close to the ideal case. Table 3

lists the values of average number of hops for few random topologies generated by the topology generator.

Table 2: Average number of hops with number of nodes increasing from 10 to 20

Nodes	Edges	Average # of Hops		
		Normal routing table	Alternate link table	Recomputed routing table
11	22	1.69091	1.77231	1.7314
12	24	1.74242	1.83144	1.7822
13	26	1.84615	1.92308	1.88264
14	28	1.87912	1.95938	1.91758
15	30	1.89524	1.97333	1.93778
16	32	1.975	2.06667	2.01198
17	34	2.02206	2.10143	2.05688
18	36	2.11	2.19181	2.14633
19	38	2.05848	2.13666	2.09234
20	40	2.17895	2.25678	2.21632

Table 3: Average number of hops for few random topologies generated by BRITE topology generator

Nodes	Edges	Average # of Hops		
		Normal routing table	Alternate link table	Recomputed routing table
4	5	1.167	1.4	1.33
5	8	1.2	1.337	1.3
7	7	2.0	3.04762	2.667
7	14	1.33	1.42007	1.38776
8	9	2.03571	2.48413	2.34921
9	11	2.13889	2.41919	2.34848
9	18	1.5	1.59	1.54
10	20	1.62	1.723	1.665
13	26	1.80769	1.89275	1.84911
15	30	1.867	1.94683	1.90571
17	34	2.02	2.10056	2.0566
20	40	2.121	2.19428	2.15382

3.3 Worst Case Difference

Though the average number of hops for alternate link table case is close to the ideal case of the recomputed routing table, the worst case difference of the path length between both the cases is also of interest. Table 4 shows the table with values of the worst case path length difference in terms of 'Y/X' and 'Y-X', where 'Y' is the average number of hops in the alternate link table case and 'X' is the average number of hops in recomputed routing table case. The table shows that the worst-case path length is increased by 200% to 300% by alternate routing. Most of these worst-cases occur between the neighbors of the failed link, i.e when source and destination are neighbors.

Table 4: Worst case path length comparison between alternate link tables and recomputed routing

Nodes	Edges	Worst case difference	
		Y/X	Y-X
11	22	2	2
12	24	2.5	3
13	26	2.5	3
14	28	2.5	3
15	30	2.5	3
16	32	2.5	3
17	34	3	4
18	36	2.5	3
19	38	2.667	5
20	40	3	4

Table 5 lists the maximum and minimum values of average number of hops with alternate link table and recomputed routing tables. The average number of hops over all (source, destination) pairs is computed for every single link failure case in the network and then the maximum and the minimum values are recorded. We can notice that the maximum and the minimum values of average number of hops in alternate routing table are quite close to the ideal case of recomputed routing tables.

4 Discussion & Related Work

The proposed deflection algorithm can be performed at different levels of the network hierarchy just as routing is performed hierarchically. For backbone networks, it can be used both at intra-pop network level and inter-pop network level where each Points of Presence (POP) [9] can be considered as a single node. The problem gets simplified when there are multiple links between the POPs or when in a POP, the routers are fully interconnected. These seem to be typical in existing backbone networks. When

Table 5: Min. and Max. values of average number of hops for alternate link tables and recomputed routing tables

Nodes	Edges	Average # of Hops				
		Normal routing table	Min. Alternate routing	Max. Alternate routing	Min. (Recomputed routing table)	Max. (Recomputed routing table)
10	20	1.64	1.66667	1.77778	1.66667	1.75556
15	30	1.88571	1.89524	2.17619	1.89524	1.98095
20	40	2.14211	2.15263	2.37368	2.15263	2.21053
25	50	2.28333	2.28667	2.47667	2.28667	2.37333
30	60	2.48736	2.49195	2.65632	2.49195	2.54943
35	70	2.50588	2.51092	2.69076	2.51092	2.56134
40	80	2.54103	2.54359	2.675	2.54359	2.61923
20	40	2.08947	2.10789	2.32368	2.09474	2.20526
20	60	1.81053	1.81579	1.92895	1.81579	1.86316
20	80	1.58947	1.59474	1.65263	1.59474	1.62105

there are multiple independent links between the POPs, then each link can act as an alternate link for other links. Within a POP, since routers are fully interconnected, deflection becomes very simple.

Our algorithm can be modified to consider link weights and link utilization to get more optimal and efficient alternate link maps. This will be part of our future work. Link overloads after re-computation of routing tables has been recently addressed in [10]. Our algorithm can be combined with such flow-based deflection algorithms to improve the link load during deflection.

4.1 Related Work

Our work is motivated by the measurements done on SPRINT network [2] and QWEST network [3]. The problem of packet drops during the transient period after a link failure is presented in [2] and [3]. The impact on voice service quality due to link failures is presented in [1]. Link failures and routing instabilities after link failures were found to be the major source of impact on the quality and availability of voice service.

Wang and Crowcroft in [11] have proposed SPF-EE (Shortest Path First with Emergency Exits) algorithm where alternate paths (APs) are provided as emergency exits in the presence of congestion or resource failures. In SPF-EE, the routing table is expanded to include two entries, one corresponding to the normal next-hop on the shortest path (SP) and other corresponding to the alternative next-hop.

If a different alternative next-hop is not available for a particular destination, the node communicates with other nodes to set up a Reverse Alternate Path (RAP) when needed. The packets are then source routed over the RAP. This approach doubles the routing table size. Also, setting up RAP will require certain time and during this time packets could get accumulated in the buffers and get dropped. Our work differs from SPF-EE by the fact that alternate paths are decided based on just the failed link and not the destination of the packet. Our approach does not increase the routing table size and does not require source routing as required by SPF-EE.

S. Iyer et al. in [10] have suggested an approach to alleviate link overload in an IP backbone by deflecting the packets to other nodes depending on their cost to destination. Their approach deflects only a fraction of the packets on an overloaded link and does not address packet reordering issues. Our approach deflects all the packets headed for the failed link and can be combined with their approach to improve service during a failure transient and after re-computation of routing tables due to a link failure.

Alternate routing using IP-in-IP encapsulation is proposed for differentiated service networks in [12]. Other previous work on alternate routing considered either a connection-oriented framework or all-optical networks, or special network topologies. Deflection routing in the form of Dynamic Alternate Routing (DAR) was proposed by Kelly et al. [13] for circuit switched networks. A Virtual Circuit Deflection Protocol (VCD) [14] was proposed by E. Varvarigos et al. where deflection is done per session basis rather than packet basis. M. Kodialam et al. in [15] have proposed a dynamic routing algorithm which can locally restore bandwidth guarantees after a single link or node failure in MPLS (Multi Protocol Label Switching) framework. In MPLS, features like standby secondary LSP and Fast Reroute [16, 17] exists for handling LSP redundancy and to protect traffic from circuit failures. Deflection routing has been proposed for all-optical networks [18, 19, 20, 21, 22] and networks with specific topological nature in [23, 24, 25].

5 Conclusions and Future Work

This paper presented a mechanism to deflect packets during the transient period, after a link failure when the routing tables are not yet recomputed. An algorithm is presented to deflect packets without causing routing loops. The presented algorithm makes minimal modifications to normal forwarding of packets. The presented algorithm does not require global knowledge of the link failure and can deflect packets around the failure quickly. The deflection algorithm continues to employ normal forwarding of packets and does not require any source routing of packets. Our evaluation has shown that the performance of the deflection algorithm is quite close to the performance when routing tables are recomputed around

the failure. The proposed mechanism could improve the network's availability of service.

In our future work, we will consider the link weight and link utilization while setting up the alternate link maps. Our future work would also involve designing more intelligent heuristics to get more optimal alternate paths. We also aim to study multiple link failures and node failures.

References

[1] C. Boutremans, G. Iannaccone and C. Diot, "Impact of link failures on VoIP performance," in *Proceedings of NOSSDAV'02*, Maimi, Florida, 2002.

[2] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya and C. Diot, "Analysis of link failures in an IP backbone," in *Proceedings of ACM Sigcomm Internet Measurement Workshop*, 2002.

[3] C. Alaettinogly and S. Casner, "ISIS routing on the Qwest backbone: A recipe for subsecond ISIS convergence," *NANOG24*, February 2002.

[4] A. Shaikh and A. Greenberg, "Experience in Black-box OSPF Measurement," in *Proceedings of ACM Sigcomm Internet Measurement Workshop*, 2001.

[5] Alberto Medina, Anukool Lakhina, Ibrahim Matta and John Byers, *BRITE Topology Generator*, Available online at http://www.cs.bu.edu/brite.

[6] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, MASCOTS'01*, Cincinnati, Ohio, August 2001.

[7] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker and W. Willinger, "Network Topology Generators: Degree-Based vs. Structural," in *Proceedings of ACM SIGCOMM'02*, Pittsburgh, August 2002.

[8] B. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, December 1988.

[9] S. Bhattacharyya, C. Diot, J. Jetcheva and N. Taft, "Pop-Level and Access-Link-Level Traffic Dynamics in a Tier-1 POP," in *Proceedings of ACM Sigcomm Internet Measurement Workshop*, 2001.

[10] S. Iyer, S. Bhattacharyya, N. Taft, N. McKeown and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," Sprint ATL Technical Report, TR02-ATL-071127.

[11] Z. Wang and J. Crowcroft, "Shortest Path First with Emergency Exists," in *Proceedings of ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Philadelphia, September 1990, pp. 166-176.

[12] S.D. Patek, Raja Venkateswaran and J. Liebeherr, "Simple Alternate Routing for Differentiated Services Networks," Technical Report, University of Virginia, CS-2000-25.

[13] R.J. Gibbens, F.P. Kelly and P.B. Key, "Dynamic Alternative Routing . Modelling and Behaviour," in *Proceedings of the 12th International Teletraffic Congress*, June 1988.

[14] E.A. Varvarigos and J.P. Lang, "A Virtual Circuit Deflection Protocol," *IEEE/ACM Transactions of Networking*, vol. 7, no. 3, June 1999.

[15] M. Kodialam and T.V. Lakshman, "Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information," in *Proceedings of IEEE Infocom*, 2001.

[16] Der-Hwa Gan, Ping Pan, Arthi Ayyangar and Kireeti Kompella, "A Method for MPLS LSP Fast-Reroute Using RSVP Detours," Internet Draft, October 2001, http://www.cs.columbia.edu/ pingpan/papers/draft-gan-fast-reroute-00.txt.

[17] Dimitry Haskin and Ram Krishnan, "A Method for Setting an Alternate Label-Switched Paths to Handle Fast Reroute," Internet Draft, November 2000, http://www.netzmafia.de/rfc/internet-drafts/draft-haskin-mpls-fast-reroute-05.txt.

[18] F. Borgonovo, L. Fratta, and J. Bannister, "Unslotted deflection routing in all-optical networks," in *Proceedings of IEEE GLOBECOM'93*, vol. 1, Houston, Texas, November 1993, pp. 119-125.

[19] F. Borgonovo, L. Fratta and J. Bannister, "Design of optical deflection-routing networks," in *Proceedings of IEEE INFOCOM'94*, vol. 1, Toronto, Ontario, Canada, June 1994, pp. 120-129.

[20] T.H. Szymanski, "An analysis of hot-potato routing in a fiber optic packet switched hypercube," in *Proceedings of IEEE INFOCOM'90*, vol. 1, San Francisco, California, June 1990, pp. 192-199.

[21] G. Castanon, L. Tancevski, and L. Tamil, "Routing in All-Optical Packet Switched Irregular Mesh Networks," in *Proceedings of IEEE Globecom'99*, Rio de Janeiro, Brazil, December 1999, pp. 1017-1022.

[22] F. Forgheri, A. Bononi, and P.R. Prucnal, "Analysis and Comparison of Hot-Potato and Single-Buffer Deflection Routing in Very High Bit rate Optical Mesh Networks," *IEEE Transactions on Communications*, vol. 43, no. 1, pp. 88-98, January 1995.

[23] A.K. Choudhury and V.O.K. Li, "Performance analysis of deflection routing in the Manhattan Street Network," in *Proceedings of IEEE ICC'91*, Denver, Colorado, June 1991, pp. 1707-1711.

[24] C. Fang and T.H. Szymanski, "An analysis of deflection routing in multidimensional regular mesh networks," in *Proceedings of IEEE INFOCOM'91*, vol. 2, Bal Harbour, Florida, April 1991, pp. 859-868.

[25] A.G. Greenberg and B. Hajek, "Deflection routing in hypercube networks," *IEEE Transactions on Communications*, vol. 40, pp. 1070-1081, June 1992.