A Fast Rerouting Scheme for OSPF/IS-IS Networks

Yong Liu, A. L. Narasimha Reddy
ELEN Department, TAMU, College Station, TX 77843
Email: {yongliu,reddy}@ee.tamu.edu

***Abstract*— Most current backbone networks use Link-State protocol, OSPF or IS-IS, as their intra-domain routing protocol. Link-State protocols perform global routing table update to route around failures. It usually takes seconds. As real-time applications like VoIP emerge in recent years, there is a requirement for a *Fast Rerouting* mechanism to route around failures before all routers on the network update their routing tables. In addition, *Fast Rerouting* is more appropriate than global routing table update when failures are transient.**

In this paper, we propose such a *Fast Rerouting* extension for Link-state protocols. In our approach, when a link fails, the affected traffic is rerouted along a pre-computed Rerouting Path. In case rerouting cannot be done locally, the local router will signal minimal number of upstream routers to setup the Rerouting Path for rerouting. We propose algorithms that simplify the rerouting operation and the Rerouting Path setup. With a simple extension to the current Link State protocols, our scheme can route around failures faster and involves minimal number of routers for rerouting.

I. Introduction

VoIP and other real-time applications require uninterrupted service from the network. It is shown that the current backbone network can deliver PSTN quality voice service with regard to delay and loss during normal condition [1]. However, it also shows that link and router failures can significantly impact a VoIP service, though infrequently.

Most current backbone ISPs use Link-State protocol, OSPF [2] or IS-IS [3], as their Intra-domain Routing Protocol (i.e. IGP, Interior Gateway Protocol). When a link fails the adjacent routers flood Link State Advertisements (LSAs) through the network notifying the failure. Routers recalculate their routing tables to route around the failure. We call this procedure as *IGP convergence* that usually takes seconds. During IGP convergence, packets originally routed along the link are likely to be dropped by the adjacent routers or by other routers because of transient routing loops.

To minimize the detrimental affect of link/node failures to real-time applications like VoIP, there is a requirement to provide a rerouting mechanism during IGP convergence. Compared to traditional IGP rerouting, i.e. LSA flooding and global recalculation of routing tables, we call rerouting during IGP convergence as *Fast Rerouting*.

In this paper, we propose a Fast Rerouting extension for Link State IGP protocols. By exploiting properties of shortest path trees, our approach has achieved a new tradeoff between complexity and response time to link failures.

II. Related work

Significant work has been done to reduce IGP convergence time. Some recent work shows sub-second IGP convergence can be achieved by utilizing layer 2 protection timer and fine tuning parameters of IGP protocols [5]. However, it is also shown that the convergence time may become a few seconds in some situation where a large amount of FIB (Forwarding Information Base) updates is required. Thus a Fast Rerouting mechanism is desirable to further shorten the time to route around failures. Moreover, global routing table update is not desirable for transient link failures. It is shown that about half of the unplanned failures last less than a minute in backbone networks [4].

MPLS based approaches, such as [6], use pre-computed backup paths to route around failures immediately after detection of link failures. However, this is usually done in a centralized manner and is not suitable for protection of all links in the network.

Failure Insensitive Routing [7], uses interface specific forwarding tables to perform fast rerouting. It requires a new algorithm for forwarding table calculation. In contrast, we propose a simpler extension to current Link State protocols. Our approach can response to link failures as soon as FIR in most cases, while in other cases the response time is reasonable.

Narváez *et al.* [8] propose a local restoration algorithm for link state protocols. The algorithm requires routers on a restoration path to change the weights of links on the path to zero and recalculate their routing tables. The calculation of routing table

and the update of forwarding tables increase the response time of the algorithm to link failures and increases the work load of the router. In contrast, our scheme calculates rerouting paths beforehand and does not need to update forwarding tables, thus has faster response to link failures.

One advantage of the above two approaches is that they do not require data plane changes. Our approach needs to modify the data plane, but the added overhead is not high.

III. Fast Rerouting Scheme

In this section we first briefly describe how our scheme works, then give algorithms used in each step. Notations used in the algorithms are listed in Table I.

A. Overview

In this paper, we assume that all links are point to point, bi-directional and with equal weights on both directions, which is generally true for backbone networks. We also assume there is at most one link failure at a time. This is because individual link failures account for nearly 70% of all unplanned failures [4] and rerouting around multiple failures requires more complex algorithms. Instead of using a complex algorithm, our scheme relies on the original IGP mechanism to handle multiple failures.

Our scheme uses the *nearest Feasible Next Hop* (with regard to number of hops) for Fast Rerouting. *Feasible Next Hop* (FNH) is defined as a router whose paths to the destinations of affected traffic do not include the failed link. When there is more than one nearest FNH, our scheme chooses the one with minimum distance to affected destinations. We define *Rerouting Path* (RP) as the path from the router adjacent to the failed link to the selected FNH. We also define the node one hop away from the nearest FNH as the *exit* node for rerouted packets.

For example, in Figure 1, a may have e, f, h, i as its FNHs for traffic affected by the failure of link (a, b) and path (a, e, h), etc. as the corresponding RPs. Our scheme will choose one from (a, f) and (a, i) that has shorter distance to b as the RP.

There are tow types of RPs: 1). Local RPs, i.e. direct FNHs; 2). RPs with one or more signaling hops. If the nearest FNH is type 1, rerouting is done locally. For example, in Figure 1, a can forward all traffic affected by link (a, b) via FNH f. If the nearest FNH is type 2, local router should setup the RP by notifying (signaling) routers on the RP about the failure and the RP before rerouting. For example, in Figure 1, a needs to notify g about the failure of (a, b) and the RP, (a, g, h). g will route all traffic affected by the failure of (a, b) to h. Since RPs are usually very short, as shown in Section IV, the cost to setup a RP is not significant.

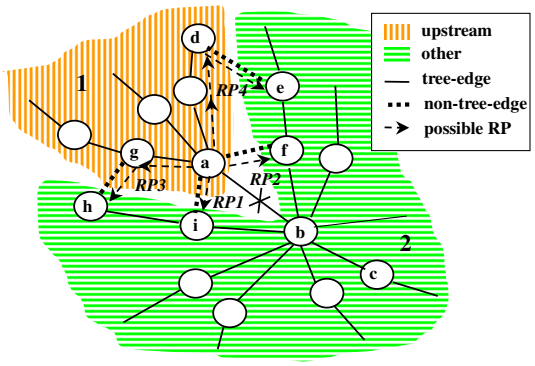

Fig. 1. The sink tree of b

B. Calculation of Rerouting Paths

In our scheme, each router calculates a RP based on its sink tree for each of its links on behalf of its neighbor. Its neighbor will use the RP to send affected traffic when the link fails. This choice of RP calculation requires routers to exchange RP information but can avoid routers to calculate sink tree for all its neighbors.

Without losing generality, we describe the algorithm for a router, b, to calculate a RP for a link between itself and one of its neighbors, a, as shown in Figure 1.

In the sink tree of b, we use BFS (Breadth First Search) in the sub-tree rooted at a to search for a RP for a and link (a, b). We call this sub-tree as the *upstream* area for link (a, b). During the search, at each node we check every neighbor of it. If the neighbor is not an upstream router, it is a FNH. As mentioned before, our scheme choose the *nearest FNH* for rerouting. When there are ECMPs (Equal Cost Multi-Paths) between a and the *exit* node we use all of them as part of the RP. We can always find a RP for a given link as long as the network is not partitioned. (See [10] for the algorithm.)

Using this method we actually find a rerouting path for traffic with destination of b. But as Theorem 1 shows, this type of RPs can be safely used for all traffic originally forwarded to b by a.

Theorem 1: Once a packet originally forwarded from a to b is rerouted to a router outside the upstream area (the 1st part in Figure 1), the packet will be routed along a shortest path without link (a, b) to its destination.

Proof: It is obvious for packets that have b as its destination.

For a packet heading for a router below b , say c, it is also true. This can be proved as below: As shown in Figure 1, the shortest path from a nearest FNH, e, to b, (e, f, b), must be shorter than the shortest path from it via a to b, (e, a, b). So the shortest path from e to a to b to c, (e, a, b, c), is longer than the shortest path from e to b to c, (e, f, b, c), i.e. the shortest path from e to a to b to c is not the shortest path from e to c. Therefore, the shortest path from e to c must not include edge (a, b). And it is obvious that the path from e to c must not include edge (b, a). ■

C. Identification of Affected Traffic

In case there is no local RP, a router on the RP needs to efficiently identify traffic affected by a link failure.

Because there are ECMPs, when a link fails, it is also possible that a destination is not reachable via one next hop but reachable via another next hop. So our scheme decides whether a destination is reachable via a given next hop.

In our scheme, a router identifies affected traffic using a simple range checking. It relies on an algorithm that assigns sequence numbers for all nodes in each *first level subtree* (i.e. the subtree under a first level child node) of the local routing tree. Sequence numbers for each subtree are independent. For each subtree, the sequence numbers are from 0 to the number of nodes in the subtree. The algorithm ensures: the sequence numbers of all nodes affected by a node failure and the sequence number of the failed node itself are continuous and thus can be represented as a simple range. In other words, when a node fails, only the nodes with sequence numbers within the *affected range* become unreachable from the root of the subtree. The start of the *affected range* of a node is its sequence number. The end of the *affected range* is the largest sequence number of all nodes affected by this node. We call the end of the *affected range* of a node as the *seq_end* number. We store the sequence number and the *seq_end* number along with each node in the subtree.

A link failure either causes the downstream node of the link to become unreachable from the root of the subtree or does not affect reachability of any destination. So the destinations affected by a link failure can also be identified using above sequence numbers. We will discuss this further in Section III-D.

For a subtree without ECMPs, we can use DFS (Depth First Search) traversal order as the sequence numbers. This is because: in a tree without ECMPs, all descendant nodes are the nodes affected by this node; and they are traversed during the period when this node is traversed.

However, for a subtree with ECMPs, if a node becomes unreachable from the root of the subtree, some of its descendants may be still reachable, because there may be more than one path from the root to the later. We have developed a modified DFS algorithm (Algorithm 1) to assign sequence numbers for nodes in a general subtree (with or without ECMPs).

TABLE I
NOTATIONS

(a, b):	link from a to b
$RP(a, b)$:	Rerouting Path of link (a, b)
T_s:	routing tree of s
$ST(T_s, i)$:	subtree of T_s rooted at i
$P(T_s, n)$:	set of parents of n in T_s
$C(T_s, n)$:	set of children of n in T_s

Algorithm 1 can be described as follows: during the DFS traversal, whenever we encounter a child node having more than one parent, we find the nearest single upstream node whose failure will cause the child node unreachable from the root. We call this upstream node as the *nearest ancestor* of the child node. (The algorithm to find the *nearest ancestor* is a reverse DFS search from the node to the root. See [10] for details.) We enqueue the child node to the *deferred DFS queue* of its *nearest ancestor*. After that we continue the DFS search for other children. After searching all its children, we call this modified DFS algorithm for the nodes in the local deferred DFS queue one by one. Similar to subtree without ECMPs, the sequence number of each node is its traversal order. Figure 2 shows the result of Algorithm 1 on a simple topology.

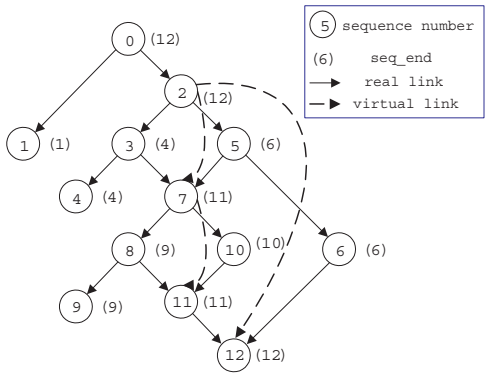

Fig. 2. Sequence numbers of nodes in a subtree tree

As Theorem 2 shows, the sequence numbers and the checking ranges assigned by the above modified

Algorithm 1:

SEQ($ST(T_s, i)$)
begin
  $ST(T_s, i).seq\_count \Leftarrow 0$
  foreach $n \in ST(T_s, i)$ **do**
    $n.enqueued \Leftarrow False$
    $n.visited \Leftarrow False$
  $DFS\_seq(ST(T_s, i), i)$
end

DFS_seq($ST(T_s, i)$, n)
/*n: DFS start node */
begin
  $n.visited \Leftarrow True$
  if $|P(ST(T_s, i), n)| > 1$ *and* $n.enqueued = False$ **then**
    $p \Leftarrow DFS\_nearest\_ancestor(ST(T_s, i), n)$
    $enqueue(p.deferred\_dfs\_queue, n)$
    $n.enqueued \Leftarrow True$
  else
    $n.seq[i] \Leftarrow ST(T_s, i).seq\_count$
    $ST(T_s, i).seq\_count \Leftarrow ST(T_s, i).seq\_count + 1$
    foreach $m \in C(ST(T_s, i), n)$ **do**
      if $m.visited = False$ **then**
        $DFS\_seq(ST(T_s, i), m)$
    while $n.deferred\_dfs\_queue \neq \emptyset$ **do**
      $m \Leftarrow dequeue(n.deferred\_dfs\_queue)$
      $DFS\_seq(ST(T_s, i), m)$
    $n.seq\_end[i] \Leftarrow ST(T_s, i).seq\_count - 1$
end

DFS algorithm can be used to identify the unreachable nodes after a node fails.

Theorem 2: In a routing tree, where each node is assigned a sequence number and an affected range using Algorithm 1, if a node fails, all and only the nodes with sequence numbers in the affected range of the node are affected.

Proof: (Summary, see [10] for details.) According to the procedure the sequence numbers are assigned by the algorithm, we need only to prove that our algorithm transforms the routing tree with ECMPs (i.e. a DAG, Directed Acyclic Graph) into a simple tree where the descendants of a node are the nodes affected by this node and the search procedure is equivalent to a DFS search in the transformed tree. Therefore, this theorem is proved according to the property of DFS. ■

D. Rerouting Operations and Setup of RP

In this subsection we give detailed description of the rerouting operations of routers after a link failure detected. The operations of routers are described as Algorithm 2 to 4.

Algorithm 2: RP_SETUP1((a, b))

/*called by a for failed link (a, b) */
$interface(b).failure \Leftarrow True$
if $interface(b).local\_reroue = False$ **then**
  foreach $RP \in RPs(a, b)$
  /*There are multiple RPs only when there are ECMPs between a and the exit node. */
  do
    $msg.RP \Leftarrow RP$
    $msg.failed\_link \Leftarrow (a, b)$
    foreach $i = RP.num\_of\_hops, \cdots, 1$
    do
      $msg.position \Leftarrow i$
      SEND_MSG($RP.nodes[i], msg$)
$reroute\_next\_hop \Leftarrow interface(b).reroute\_next\_hop$
$rerouting \Leftarrow True$

1) Operations of local router (Algorithm 2): After detection of a link failure, the local router first marks the next hop unreachable. As a result, for traffic having other ECMP next hops, the router will avoid using this next hop. If the RP for the failed link is a local FNH, the router set the *rerouting flag* and the *rerouting next hop*. If the RP includes some upstream nodes, the router send messages to them to notify the link failure and the RP before rerouting traffic along the RP.

2) Operations of routers on the RP (Algorithm 3): When a router receives a RP setup request, it marks *affected* interfaces and sets the rerouting flags and affected range for the interface. We call an interface *affected* when some packets forwarded through the interface cannot reach their destinations because of the link failure.

3) Modified forwarding operations (Algorithm 4): After the RP is setup, the router adjacent to the failed link will reroute all traffic routed to the failed link along the RP; other routers on the RP check traffic to be forwarded via affected interface and reroute affected traffic along the RP as shown in Algorithm 4.

Theorem 3 ensures the correctness of Algorithm 3 and 4.

Theorem 3: For a router on the RP, when link (a, b) fails, if and only if (a, b) is an edge of the

Algorithm 3: RP_SETUP2(msg)

```
/*called by a router on the RP, n */
(a, b) ⇐ msg.failed_link
foreach m ∈ C(T_n, n) do
    f ⇐ interface(m)
    if (a, b) ∈ ST(T_n, m) and
    b.seq[f] ∈ [a.seq[f], a.seq_end[f]] then
        f.affected ⇐ True
        f.affected_start ⇐ b.seq[f]
        f.affected_end ⇐ b.seq_end[f]
if msg.RP.num_of_hops > msg.position
then
    reroute_next_hop ⇐
    msg.RP.nodes[msg.position + 1]
else
    reroute_next_hop ⇐ msg.RP.next_hop
rerouting ⇐ True
```

Algorithm 4: NEW_FWD(pkt)

```
(dest_node, next_hops) ⇐
routing_table_lookup(pkt)
if rerouting = False then
    normal_forward(next_hops, pkt)
else
    valid_next_hops ⇐ ∅
    foreach n ∈ next_hops do
        f ⇐ interface(n)
        if f.affected = False then
            valid_next_hops ⇐
            valid_next_hops ∪ n
        else if dest_node.seq[f] ∉
        [f.affected_start, f.affected_end]
        then
            valid_next_hops ⇐
            valid_next_hops ∪ n
    if valid_next_hops ≠ ∅ then
        normal_forward(valid_next_hops, pkt)
    else
        reroute(reroute_next_hop, pkt)
```

first level subtree below an interface (assume a is the upstream node of the link) and the sequence number of b is within the affected range of a for the interface, then b becomes unreachable via that interface.

Proof: If b is within the affected range of a, then the failure of a causes b unreachable, i.e. the traffic to b must pass a. Because our scheme uses the nearest FNH to reroute, a does not have ECMPs to b, otherwise a will reroute locally. So the failure of link (a, b) causes b unreachable via the interface.

If link (a, b) is not an edge of the subtree, no traffic via the interface will pass the link. If link (a, b) is an edge of the subtree but the sequence number of b is not within the affected range of a, then the failure of a will not affect traffic to b, i.e. there is a ECMP not including link (a, b) from the root of the subtree to b. ■

IV. Evaluation

We have evaluated our rerouting scheme on random topologies generated using BRITE topology generator [9]. We have generated topologies of 25-200 nodes with average degree of 4, 6 and 8. The link weights are uniformly distributed between 100 and 300. For each configuration we have generated 5 random topologies.

A. Number of Signaling Hops

First, we have measured the number of signaling hops of rerouting paths. (0 means local rerouting, 1 means the *exit* node is 1 hop away, and so on)

As shown in Figure 3, the maximum number of signaling hops is 2 for all topologies we used. For topologies with average degree of 6 and 8, most RPs are local FNH.

The small value of number of signaling hops is beneficial for our rerouting scheme, since it means short RP setup time, i.e. the response time of our rerouting scheme. For signaling hops of 0, the response time of our scheme is near zero. For signaling hops of n, the response time of our scheme is the time it takes to send a message to a router n hops away in the network plus the processing time of routers.

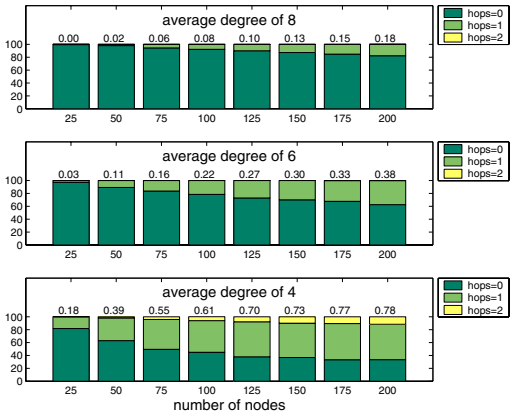

Fig. 3. Percentage of RPs with different number of signaling hops and average number of signaling hops (the number above each vertical bar)

B. Path Elongation

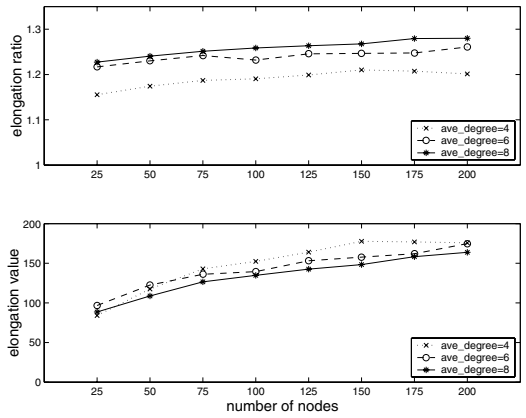

Fig. 4. Elongation ratio and value compared to optimal paths

We have measured the distance elongation of our rerouting scheme compared to the optimal shortest path routing.

We define the *elongation ratio* (*elongation value*) as the ratio (difference) of the distance between an affected pair of nodes under our rerouting scheme and the optimal distance after global routing table recalculation.

Since our main objectives are to achieve fast response to link failures and simple rerouting operations, our scheme does not prioritize the optimization of the rerouting path. But as we see in Figure 4, the path elongation is not significant. While the average elongation ratio is about 1.2[1], the elongation value value is about 100 to 160, i.e. less than twice of the minimum link weight, 100. This means the average elongation of our scheme is less than two hops.

C. Complexity of Algorithms

The most complex algorithm in our scheme is Algorithm 1 that calculates sequence numbers for all nodes in each first level subtree of the local routing tree.

In a routing tree without ECMPs, the complexity is same as DFS, i.e. $O(V)$ where V is the number of nodes in the routing tree.

In a routing tree with ECMPs the complexity can be estimated as $O(nV + nV'E')$, where n is the maximum degree of the root node, V' is the maximum number of nodes in a first level subtree that have more then one parent, E' is the maximum number of unique edges traversed in a call of $DFS\_nearest\_ancestor$, which is at most V. $O(nV)$ represents the complexity of the DFS search procedures for all first level subtrees. $O(nV'E')$ represents the complexity of the calculation of all *nearest ancestors*. So a loose upper bound of the whole algorithm is $O(nV^2)$. But in real-world network topologies the complexity is much smaller (see [10]). Besides, using incremental implementation by storing the $deferred\_dfs\_queue$ with the nodes in first level subtrees, the complexity of this algorithm can be further reduced.

[1]The elongation ratio is decided by the elongation value and the average distance between pairs. While the increase of average degree reduces the elongation value, it also reduces the average distance. And we can see it increases the elongation ratio as shown in Figure 4

V. Future work

First, we plan to use more than one RPs to split rerouted traffic for load balancing. Second, we plan to enhance our algorithm to detect multiple link failures and node failures. In such cases we should avoid fast rerouting and rely on the original IGP convergence mechanism.

VI. Conclusions

We have proposed a Fast Rerouting scheme for OSPF/IS-IS networks in this paper. We have developed efficient algorithms for calculation of Rerouting Path, and identification of affected traffic. The rerouting operation for each packet is comparable to basic IP forwarding. Simulation results show that, assuming there is one link failure at a time which accounts for a large portion of network failures, our scheme achieves fast response to link failures and the path elongation compared to optimal path is not significant.

References

[1] C. Boutremans and G. Iannaccone and C. Diot, *Impact of link failures on VoIP performance*, NOSSDAV, 2002.

[2] J. Moy, *OSPF Version 2*, RFC 2328, Apr. 1998.

[3] D. Oran, *OSI IS-IS Intra-domain Routing Protocol*, RFC 1142, Feb. 1990.

[4] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, C. Diot, *Characterization of Link Failures in an IP Backbone*, INFOCOM 2004.

[5] N. Dubois, B. Fondeviole, N. Michel, *Fast convergence project*, RIPE-47 presentation, Jan. 2004.

[6] P. Pan, G. Swallow, A. Atlas, *Fast Reroute Extensions to RSVP-TE for LSP Tunnels*, Internet Draft, Feb. 2004.

[7] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, *Failure Insensitive Routing for Ensuring Service Availability*, IWQoS 2003.

[8] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, *Local Restoration Algorithm for Link-State Routing Protocols*, ICCCN 1999.

[9] A. Medina, A. Lakhina, I. Matta, and J. Byers, *BRITE: An Approach to Universal Topology Generation*, In Proceedings of MASCOTS 2001, Aug. 2001.

[10] Y. Liu, A. L. Narasimha Reddy, A Fast Rerouting Scheme for OSPF/IS-IS Networks, technical report available at http://ece.tamu.edu/techpubs, Dept. of Electrical Engineering, Texas A&M University, 2004