A Novel Loop-Free IP Fast Reroute Algorithm

Gábor Enyedi, Gábor Rétvári, and Tibor Cinkler

High Speed Networks Laboratory,
Department of Telecomunications and Media Informatics,
Budapest University of Technology and Economics
H-1117, Magyar Tudósok körútja 2., Budapest, Hungary
{enyedi, retvari, cinkler}@tmit.bme.hu

Abstract. Although providing reliable network services is getting more and more important, currently used methods in IP networks are typically reactive and error correcting can take a long time. One of the most interesting solutions is interface based fast rerouting, where not only the destination address but also the incoming interface is taken into account during the forwarding. Unfortunately, current methods can not handle all the possible situations as they are prone to form loops and make parts of the network with no failure unavailable. In this paper we propose a new interface based routing method, which always avoids loops for the price of a bit longer paths. We also present extensive simulation results to compare current and proposed algorithms.

Keywords: IPFRR, IP, fast, reroute, routing, interface.

1 Introduction

In the last few decades Internet has become one of the world's most significant communication systems. Although fault tolerance was always one of the most important attributes of this network, using the traditional resilience mechanism of IP usually needs significant time to converge. However, convergance times should decrease. With the growing of networks the time of transients between the failure and reconfiguration is getting longer, and today this convergence can even take some minutes.

The main reason of this slow reconfiguration is the reactive approach for recovery of a failure taken by conventional routing protocols, like the Open Shorthest Path First (OSPF) or the Intermediate System-to-Intermediate System (IS-IS) routing protocol used ubiquitously in modern IP networks. Here, fault recovery is assured by recomputing the routing tables at each router after a failure shows up, which might take significant time.

In contrast to traditional IP error correction techniques, which are fundamentally reactive, the new IP Fast Reroute (IPFRR) framework proposes proactive solutions, so these methods are always ready to reroute packets. Naturally this reroute must be done *locally* because there is no time for any communication. Using these algorithms transient link errors can also be avoided; since packets can reach the destination, the starting of reconfiguration of the network can be delayed, so it can be done if the persistence of the failure is sure.

A. Pras and M. van Sinderen (Eds.): EUNICE 2007, LNCS 4606, pp. 111–119, 2007.

The simplest proactive solutions only work in some special networks. For example, Equal Cost Multiple Path (ECMP) [1] can use multiple shortest paths to a destination, if exists, and is able to shift traffic from a failed shortest path to another one, unaffected by the failure. Routers using Loop-Free Alternates (LFAs) [2], another IPFRR technique, need a neighbor with a shotest path not containing the failed resource. Improved version of this method [3] needs a reachable node from where the destination can be reached, and packets are tunneled to this new node. The first solution that promises 100% fault recovery is Not-Via addresses [4]. Here, each interface has two addresses, and the second one means that the link is down, so packets are tunneled on a path to the next hop, which bypasses the failed link.

One of the most interesting possibilities is interface based routing. These methods use both the address of the destination and the incoming interface for selecting the next hop. Using this extra information Failure Insensitive Routing (FIR) [5] does not need significantly new hardware and it can survive single link failures. Although the authors presented two algorithms, the computed routings are the same, so we refer to these as FIR in the next part of this paper.

Unfortunately, FIR can create loops when more than one link or at least one node is unavailable (a loop is a forwarding cycle, which is never left by packets, and packets are dropped when TTL is up). Loops have usually serious effects. When a loop exists in a network even those parts of the system can become unavailable, where all the resources are operable, because the high load increases the probability of losing packets. In this paper we give a possible solution of this problem, based on the observation that loops can be avoided if not always the shortest paths are used in normal operation.

The rest of this paper is organized as follows: in Section 2 we prove that FIR can create loops. In Section 3 we present a novel routing solution, which can tolerate single link failures and never creates loops. In Section 4 we discuss implementation questions and in Section 5 we compare the current and the proposed algorithms using simulation results. In the last section we summarize our results.

2 Loops Using FIR

It was mentioned previously and proved in [5] that it is possible to correct one link failure with FIR in a network using interface-based routing. First, we recall the algorithm of FIR then we show that FIR can create loops.

The base idea of FIR is simple: if a node gets a packet from a neighbor which usually does not use this direction for forwarding, then there is a failure in the network. FIR calculates which links could have been failed and also – if it is possible to avoid them – another path to the destination. Using this information, FIR precomputes an alternative route for each incoming interface which is guaranteed to avoid the failed links. It is important to mention that if all the links are available FIR uses shortest paths for forwarding.

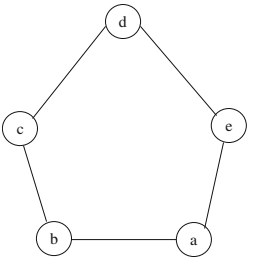

Fig. 1. A network with ring topology

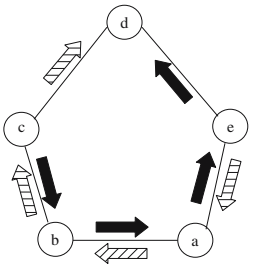

Fig. 2. The two arborescences of LFIR

Although this method is very effective it has some serious drawbacks because of the behavior when no failure exists. Theorem 1 shows this. For purposes of this paper, we call a routing *optimal* if packets are always forwarded along shortest paths if all the links are available. A routing is *failure insensitive* if traffic can pass between all nodepairs even if one link is down. We call a routing *loop-free*, if there is no loop even if any subset of links and/or nodes has failed.

Theorem 1. *There are some networks where no optimal, failure insensitive and loop-free interface-based routing exists.*

Proof. See network on Fig. 1, which uses an optimal and failure insensitive interface-based routing, and let all the lengths of the links be 1. Let link $\{e, d\}$ and $\{c, d\}$ be both unavailable. It is easy to see that packets heading from a to d will be sent to e because the routing is optimal. Naturally e will send packets on path $e - a - b - c$ because failure insensitive interface-based routing is used. Because $\{c, d\}$ is down and c is the default next hop of b – so c "thinks" $\{c, d\}$ is the first unavailable link –, c tries to correct the error and packets are sent back following path $c - b - a - e$. So a routing loop is formed between nodes e and c, which completes the proof of the theorem. □

Because FIR is interface based, optimal and failure-insensitive Theorem 1 proves, that there are some networks where using FIR can cause loops. Section 5 shows that these networks are not rare; FIR can cause loop in most networks.

3 Loop-Free Fast Interface-Based Routing

It was shown in the previous section that FIR can create loops. In this section, we propose a novel technique, Loop-Free Failure Insensitive Routing (LFIR), which can solve this problem. Naturally LFIR is not an optimal routing. First we deal with the problem of 2-edge-connected networks (networks with two edge-disjoint paths between each nodes), then we discuss an improved version of the first algorithm, where 2-edge-connectivity is not necessary.

In the rest of this paper we refer to the set of vertices of graph G as $V(G)$ and the set of edges as $E(G)$. If there is an edge between a and b we refer to this as $\{a, b\}$, if it is an undirected edge, or (a, b) if it is a directed one (a is the source, b is the target).

The basic idea of LFIR is to find paths from each node to each destination in such a way that when a node gets a packet from a specific incoming interface, it can always decide if either the default path was used or the packet is on a detour due to a failed link. If the detour has also failed, the packet must be dropped. To do this we must recall a special version of a theorem of Edmonds which comes easily from [6].

Definition 1. *A branching (spanning arborescence) rooted at vertex d in digraph G is a spanning tree directed in such a way, that each vertex $x \neq d$ has one edge going out. (Note that branchings are usually defined in the reverse direction.)*

Proposition 1. *Let G be a digraph, which is 2-edge-connected. It is possible to find two edge-disjoint branchings in this graph rooted at any $d \in V(G)$.*

One may observe that a branching is something like routing for a given destination d; if a packet can follow the directed edges of a branching rooted at d it reaches the destination. The only difficulty is that links can be used in both directions, so it can be modeled with an undirected graph.

It is possible to solve this problem. Let G be the undirected 2-edge-connected graph of a network. Let G' be a directed graph such that $V(G) = V(G')$ and if $\{i, j\} \in E(G)$ then $(i, j) \in E(G')$ and $(j, i) \in E(G')$. It can be easily proven that G' is also 2-edge-connected.

Now, the version of LFIR for 2-edge-connected networks is the following. Convert the undirected graph G to a digraph G', find two edge-disjoint branchings in G' rooted at d for all $d \in V(G')$. For each destination label the two branchings (1 and 2). When a packet arrives following a particular branching (the destination contained by the packet and the incoming interface shows which branching is that), forward it following the same one if it is possible – there is exactly one outgoing edge of a branching at each node. If it is not possible and the packet used the first branching, try to forward it following the second one; if it used the second one, drop the packet. Use the first branching at the first hop, if it is possible. The routing for d (which consists of the two branchings) is shown in Fig. 2. The next theorem shows that packets always reach the destination if at most one link is down and that loops can never be created.

Theorem 2. *The version of LFIR used in 2-edge-connected networks is correct (it never creates forwarding loop) and complete (packets arrive if at most one link is down).*

Proof. It is easy to see that packets can travel on each link at most two times – once using branching one and once using branching two –, so there can not be a forwarding loop. It is also easy to see that packets arrive to the destination along branching one if all the links are available.

Now suppose that exactly one link, $\{i, j\} \in E(G)$ is failed. Naturally $(i, j) \in E(G')$ and $(j, i) \in E(G')$ and these two edges can not belong to the same branching, because there is no cycle in branchings. Suppose that a packet can not reach the destination. First it is forwarded along the first branching. However its forwarding failed, so it was tried to use link $\{i, j\}$ which means that either (i, j) or (j, i) is in the first branching. Without loss of generality, we can suppose that this edge is (i, j). So the packet has left node i using the second branching. Failing the forwarding again means that link $\{i, j\}$ was tried to be used again, so (j, i) is an edge of the second branching. But the packet could reach node j from node i meaning that there is a path from i to j in the second branching and with (j, i) there is a cycle which contradicts the assumptation that there is no cycle in a branching. □

Next we deal with the problem of non-2-edge-connected networks. If the network is not 2-edge-connected two edge-disjoint branchings can not be found, but correcting the errors of those links, which do not cause the network to fall into two parts is still possible.

An undirected graph can be partitioned into z disjunct components, such that these components are 2-edge-connected. Naturally, it is possible that some componets contain only one vertex. If leaving a link causes the network to fall into two parts, it means that this link – a bridge – is between two 2-edge-connected components. It is also true that if vertices s and d are not in the same 2-edge-connected component, there is only one edge-disjoint path between them.

Using these ideas one may observe a possibility to improve LFIR. Duplicate the bridges virtually in the graph of the network. This new graph is 2-edge-connected, so after the transformation to a directed graph there will be at least two edge-disjoint branchings. Packets can follow these branchings as before. If a packet following a branching crosses a bridge, then the node after the bridge can not decide which branching was used, so use the first one for the next forwarding.

It can be easily proven that all the bridges are used by both branchings and each is used in the same direction in the directed graph (i.e. if $\{i, j\}$ is a bridge, then both branchings contain (i, j) or both contain (j, i)).

Theorem 3. *The improved version of LFIR is correct and complete (packet will arrive if at most one non-bridge link is down).*

Proof. If there is no failure packets follow the first branching and d (the root) is reached. If one link used by branching one is down, and it is not a bridge, packets either reach node d – if it is in the same component (because of Theorem 2) – or leave the 2-edge-connected component following the second branching, and after that they will reach d following the first branching. The algorithm is complete.

Now suppose that the algorithm is not correct. If there is a forwarding loop, there must be more than one failed link or at least one failed node, because the algorithm is complete. First suppose that all the nodes are available. In this case it is true that the forwarding loop must leave at least one 2-edge-connected component because of Theorem 2. So there must be a component – let it be component A – that packets leave and then return to it. Let 2-edge-connected

component B be the last component which was left before packet returns to A. There must be a bridge $\{i, j\}$ such that $i \in A$ and $j \in B$. Because each bridge is used in the same direction there is one (j, i) and no (i, j) edge in both branchings, so the packet did not leave A using bridge $\{i, j\}$. But his means there is a path from component A to component B without edge $\{i, j\}$, and another with $\{i, j\}$ which contradicts the assumptation that $\{i, j\}$ is a bridge.

Node failure can be treated like some link failures. If the source is available and some nodes go down it has the same effect as all the links of these nodes become unavailable, so the algorithm is correct. Naturally, if the source is down it can not send any packet. □

4 Implementation Questions

In the previous section we proposed an algorithm for constructing a loop-free failure insensitive routing. In this section, we discuss some implementation questions which are still open.

Finding branchings: For LFIR the most important is an effective algorithm for finding branchings. Note that, unlike the ones in the literature, our branchings are directed *towards* the destination, not *away* from it. However, this does not cause any problem since any algorithm described below can be twaeked to reverse the direction of the branchings found. The fastest algorithm – known by the authors – was proposed by Tarjan [7], and it needs $O(e\alpha(e, n))$ time, where $e = |E(G)|$, $n = |V(G)|$ and $\alpha(e, n)$ is a very slowly growing function related to the inverse of Ackerman's function. The value of this function is practically a constant. Although these methods are very fast, we have used an algorithm of Lovász [8]. This algorithm is simple and fast enough for our purposes (it takes only $O(e^2)$ steps to find two branchings with breadth first search). But more importantly, Lovász's algorithm allows us to apply a quick heuristic to decrease the length of the paths in the primary branching (used as the default path, i.e., when there are no errors): we always choose the directed edge from the set of edges that can be added to the arborescence, which provides the shortest path to the target of this edge. Using binary heap with this heuristic $O(e^2 \log e)$ time is needed.

Finding bridges: If it is not sure that the network is at least 2-edge-connected, it is needed to find the bridges. Finding bridges can be done in $O(e\alpha(e, n))$ [7] time, but this algorithm is complicated. We used to check if all the nodes are reachable after leaving an edge with a breadth first search. If not, the selected link is a bridge. This needs $O(e^2)$ time.

Using LFIR in distributed environment: Using LFIR in distributed environment – such as routers in a network – raise a new problem; routers must find the same two branchings. A unique priority given to all the edges can solve this problem. If there are more edges with the same distance from the root during the edge selection, choose always the one with the highest priority. In this way building

a branching is fully defined, so if the routers have the same information about the network the same routing will be calculated.

5 Simulation Results

In the previous sections algorithm LFIR was presented. Although it was discussed that using this algorithm loops can be avoided for the price of using longer paths in normal operation, the probability of loops using FIR and the lengths of path using LFIR are still unknown. In this section, we answer these two questions.

During all the simulations we used the topology of real networks – the NSF network [9] and the backbone network of Germany and Italy [10]. To make these networks random we used random edge lengths; the distribution of lengths was independent, discrete and uniform between 1 and 50. We presumed that FIR drops packets only if it can't forward them (edge of detour is down).

Table 1. Average probability of loops when two edges or one node is down

Top.	Prob. of loops w/ failed edges	Prob. of loops w/ failed node
NSF	5.37 %	74.45 %
Germany	10.04 %	90.62 %
Italy	4.2 %	83.5 %

Table 2. Average path lengths using LFIR related to using shortest paths

Top.	LFIR w/ heur.	LFIR w/o heur.
NSF	106.27 %	137.37 %
Germany	116.36 %	146.15 %
Italy	112.07 %	150.38 %

In the first and second simulations we studied the possibility of loops in networks. We used random experiments to decide if it is possible to remove exactly two links – first simulation – or exactly one node – second simulation – from the random network such a way that FIR makes loop. These experiments can be modeled by a Bernoulli random variable, so it can be proven using Chebyshev's inequality that after 250000 experiments a symmetrical confidence interval with size 0.02 (the difference between the real probability and the approximated is at most 1%) at level 99% can be calculated.

The result of these simulations is surprising: it was always possible to create loop with FIR in all the studied topologies irrespectively of the given lengths of edges – all the probabilities were 100%. Naturally this means only that network topologies in which FIR is prone to forming routing loops are quite common, not that FIR can create loop in all networks.

In the third and fourth simulations, we studied the probability that loops show up when using FIR. Therefore, we conducted another experiment to decide if there is a loop in a given random network if two randomly selected links or one randomly selected node is down. The confidence interval is the same as previous.

The result of these simulations is presented in Table 1. Curiously, the probability of FIR forming loops is not negligible, above all if it is a node that fails $(75 - 90\%)$ and not a link $(4 - 10\%)$. We believe that this experiment evidences

that, without clever modifications, FIR is prone to forming routing loops in case of a multiple link or single node failure. Fortunately, LFIR is guaranteed to avoid loops at the cost of an insignificant growth in the average path lengths, as testified by the simulations in the following.

To answer the question of path lengths calculated by LFIR we made 250000 random experiments with each topology. We calculated the average path lengths to each destination using LFIR – with and without heuristic (Section 4) – and shortest paths, and the quotient of these path lengths were summed. In this way we calculated the average ratio between the two methods. Results are found on Table 2. It can be observed that the greatest increase of average length of paths is only 16 %, which is low enough to let most networks use LFIR.

6 Conclusions

In this paper, we discussed methods for interface-based fast IP rerouting, a new type of proactive routing technique insensitive to link failures. We have shown that the FIR algorithm has the disadvantage that it can create routing loops – which has usually devastating effects – when more failed links or failed nodes are present. Our simulation results show that, depending on the specifics of a network, forming such loops can become quite common. We gave a formal proof that routing loops in FIR can be attributed to the fact that it uses shortest paths when there is no failure in the network. Based on this observation, we proposed LFIR, a novel failure insensitive routing solution. LFIR basically achieves a trade-off between optimality and correctness: it guarantees loopfree error recovery while, at the same time, it increases the length of the default paths. We presented extensive simulation studies to show that this increase of path lengths is tolerable (less than 16% in average). Our results indicate LFIR can bring important benefits to almost all IP networks, where a small surplus of capacity is present to accommodate the slightly longer routes.

References

1. Thaler, D.: Multipath issues in unicast and multicast next-hop selection. Internet Engineering Task Force: RFC 2991 (November 2000)
2. Atlas, A.: Loop-free alternates for ip/ldp local protection. Internet Draft (March 2005), available online: `http://tools.ietf.org/html/draft-ietf-rtgwg-ipfrr-spec-base-00`
3. Bryant, S., Filsfils, C., Previdi, S., Shand, M.: Ip fast-reroute using tunnels. Internet Draft (April 2005), available online: `http://tools.ietf.org/html`, `http://draft-bryant-ipfrr-tunnels-02`
4. Bryant, S., Shand, M., Previdi, S.: Ip fast reroute using not-via addresses. Internet Draft (December 2006), available online:`http://www.ietf.org/internet-drafts/draft-ietf-rtgwg-ipfrr-notvia-addre sses-00.txt`
5. Nelakuditi, S., Lee, S., Yu, Y., Zhang, Z.-L., Chuah, C-N.: Fast local rerouting for handling transient link failures. Accepted for publication in IEEE/ACM Transactions on Networking (December 2006), available online: `http://arena.cse.sc.edu/papers/fir.ton.pdf`

6. Edmonds, J.: Edge-disjoint branchings. Combinatorial Algorithms, 91–96 (1973)
7. Tarjan, R.E.: Edge-disjoint spanning trees and depth-first search. lnf. Proc. Letters 3(2), 51–53 (1974)
8. Lovász, L.: On two minimax theorems in graph theory. Journal of Combinatorial Theory, 96–103 (1976)
9. Chinoy, B., Braun, H.W.: The national science foundation network. Tech. Rep., CAIDA (September 1992), available online: `http://www.caida.org/outreach/papers/1992/nsfn/nsfnet-t1-technology.pdf`
10. Garcia-Osma, M.L.: TID scenarios for advanced resilience. Tech. Rep., The NOBEL Project, Work Package 2, Activity A.2.1 (September 2005)