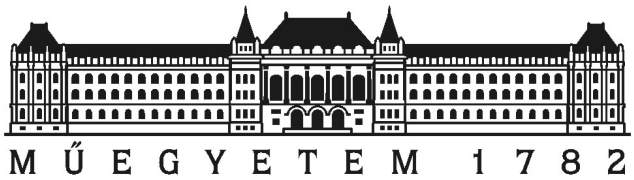


DEPARTMENT OF TELECOMMUNICATIONS AND MEDIA INFORMATICS
BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS

NOVEL ALGORITHMS FOR IP FAST REROUTE

Collection of Ph.D. Theses
by
Gábor Enyedi

Research Supervisors:

Dr. Gábor Rétvári
Department of Telecommunications and Media Informatics

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
BUDAPEST, HUNGARY
FEBRUARY 2011

1 Introduction

Our life has changed significantly during the last few decades. Thanks to the continuously increasing spreading of modern communication networks, nowadays it takes no time to reach anybody, or to find information about almost anything. Moreover, it seems that currently a common platform for all the communication networks is forming, which role is likely to be played by the Internet Protocol (IP).

Unfortunately, IP still has problems, when it needs to transport real-time traffic, like Voice over IP (e.g., in 3G, 4G mobile networks), IPTV, on-line gaming, or even business critical stock-exchange transactions. Furthermore, there are several industries, which lease Virtual Private Networks (VPN) with strict Service Level Agreements (SLA), and use it for IP telephoning or even accessing remote applications. Since IP was designed for transporting elastic traffic, where latency or the speed of recovery after a failure was not an important issue, even current IP networks have difficulties to fulfil the needed Quality of Service (QoS) requirements. My theses provide solutions to the recovery issues.

In a communication network, two main types of recovery can be applied: protection and restoration. Protection techniques are *proactive* techniques, which prepare to failures, and compute the way of bypassing a resource long before it goes down. Naturally, since preparing to all the possible combinations of failures is impossible, these techniques are moderately robust and they often use suboptimal paths. However, these techniques are extremely fast, since after a failure simply switching to a precomputed detour is needed.

In contrast, restoration techniques are *reactive*, which means that they start to find the way of overcoming a failure, after it occurred. Therefore, these techniques have complete information about the new topology, which fact makes them very robust. On the other hand, reactive behaviour makes restoration techniques much slower than protection ones are.

Unfortunately, since IP was designed for transporting pure elastic traffic, there is no native protection in these networks, but their recovery is based exclusively on restoration techniques like Open Shortest Path First (OSPF) [Moy98] or Intermediate System to Intermediate System (IS-IS) [fS02]. This lack of protection is getting more and more problematic with the spreading of IP, thus several proposals were made in order to endow IP with this important capability. These native protection techniques are the IP Fast ReRoute (IPFRR) [SB10] techniques.

Since IPFRR mechanisms are protection mechanisms, their principles significantly differ from traditional recovery: these techniques put the traffic *locally* to a *precomputed detour*. This means that when a resource goes down, its immediate neighbours detect the failure and reroute the packets without informing other nodes about the change of topology; Figure 1 illustrates this scheme.

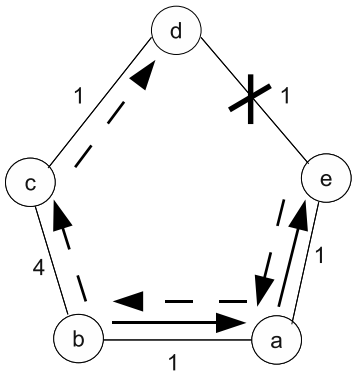


Figure 1. Example for local rerouting; the numbers next to the edges are representing their lengths

Suppose that node b has some packets to send to d. Then, the shortest path from b to d is path $b \to a \to e \to d$. However, if the link between e and d goes down, only the immediate neighbours of the link, e and d, are the ones, which detect the failure. Since there is no time to advertise the fact that the link is failed, node e locally reroutes the packets. Since there is no other possibility, node e simply sends the packets back to a. If this network uses IPFRR, then a does not send the packets to e again, and does not make a forwarding loop, instead it forwards them to b which sends them to c instead of a, and packets finally reach d. It is essential to observe that a and b still does not "know" the failure and they are not reconfigured. There are simply some entries in the Forwarding Information Bases (FIB) that tell that special packets returned from e must be forwarded in another way. Thus, while only IPFRR reroutes the packets, *all* the packets follow the suboptimal path $b \to a \to e \to a \to b \to c \to d$. Naturally, this means that in the case of a persistent failure (after some time), some recovery technique is needed, which optimizes the forwarding paths.

2 Research Objectives

Previously, the principles of IPFRR were discussed. Now, we review the main requirements IPFRR proposals must meet, then I point out the problems current IPFRR mechanisms suffer from, and finally, based on these observations, I present my research objectives.

The requirements, a modern IPFRR technique must fulfil, are as follows. First, these mechanisms are needed to be applicable inside an autonomous system, so they are considered as a supplement of standard IGP routing. Second, IPFRR techniques are needed to be extremely fast; as a rule of thumb it is usually said that they need to overcome a failure in at most 50ms, which is acceptable even for voice calls.

Moreover, for achieving this performance, the way of packet forwarding of traditional IP systems must not be modified significantly. Furthermore, as many single link or node failures must be covered, as it is possible. However, multiple failures do not need to be protected, but they should be detected in order to immediately fall back to traditional recovery. Finally, detours should be preferably as short as possible.

Naturally, not all the IPFRR techniques are worth the same, and each of them has its own drawbacks. Some of them cannot cover all the single failure cases, others are prone to create loops in the case of multiple failures, and there are some proposals, which bring significant, mostly management, overhead.

Therefore, my research objective is creating an IPFRR technique, which fulfils all the previous requirements; so this technique must always bypass any single failure, must never create loop and brings only moderate extra overhead.

3 Methodology

Below, I present algorithms, which solve some problems related to IPFRR. Therefore, the methodology of my research is principally based on the toolset of graph and algorithm theory; applying mathematics, I prove properties of such algorithms (correctness, completeness and complexity).

However, several algorithms I proposed do not only solve particular problems, but optimize the results as well by using heuristics. Although the completeness and correctness of the algorithms are proven by theoretical results, the performance of these heuristics are studied by extensive simulations.

Moreover, my work also produced a prototype implementation. Thus, in some cases, I was able to use measurements instead of simulations.

4 New Results

4.1 Multiple Failures with Interface-based IP Fast ReRoute

In order to cover 100% of single failure cases, packets are needed to be marked somehow. This marking can be explicit, when the header is modified or extended, or implicit, when some extra information is taken into consideration, which is usually not used for standard IP forwarding. Such information can be the incoming interface, which roughly indicates the path the packet has already made. As it will turn out in this section, such forwarding, called *interface-based forwarding*, which takes into consideration not only the destination address but also the incoming interface, can be well applied for IPFRR.

First, it seems that realizing interface-based forwarding is quite easy; since modern routers have linecards with dedicated memory, and since forwarding decision is made by these linecards, one may think that simply downloading different Forwarding Information Base (FIB) to the linecards would result in interface-based forwarding. Unfortunately, a typical linecard has multiple interfaces, which all must be handled differently. Therefore implementation of such technique is not easy, albeit it is undoubtedly possible without significantly new hardware.

There are several IPFRR proposals based on interface-based forwarding. The first one among these mechanisms is called U-turn Alternates [Atl06], which use this capability to detect, when a next hop sends packets back. Failure Insensitive Routing (FIR) [NLYZ03, LYN+04, NLY+07] improved this technique and it is capable to bypass any single link failure by using not only the next hop, but any incoming interface through which packets do not arrive in a failure-free case. Later, this scheme was further improved in order to cover node failures [ZNY+05] (this technique is referred as Failure Inferencing based Fast Rerouting (FIFR) in the sequel), and to bypass any single link or node failure simultaneously [WN07].

Unfortunately, all these techniques have a serious shortcoming: they are prone to form loops when multiple simultaneous failures occur. Since IPFRR can be used for avoiding reconfiguration in the case of transient failures, the time, during only IPFRR provides connectivity, can be several seconds or even a minute,[1] so loops created by IPFRR may last long. Moreover, these loops do not only delay the recovery to a level much worse than using simple restoration, but they can cause congestion as well. Even the authors of FIR have recognized the problem, and proposed Blacklist-based Interface-Specific Forwarding (BIFS) [WZN06], which is, however, not applicable in backbone networks, since this proposal needs more than simple interface-based forwarding.

I have proposed an IPFRR technique, Loop-free Failure Insensitive Routing (LFIR)[C2, C3, P1], which can always bypass a single failed link, and it can never create loops. LFIR is introduced in Thesis 1.2.

THESIS GROUP 1: [C2, C3, P1] *I have shown that detours computed by U-turn alternates, FIR and FIFR are not necessarily loop-free. I have proposed a new loop-free interface-based IP fast reroute method in order to avoid this shortcoming.*

The main reason of forming loops stems from the way of selecting default paths. If in the intact network shortest paths are used for default forwarding, it has the possibility, that network "forgets" the first failure, and tries to bypass a second one, in this way causing loops. Thesis 1.1 is based on this observation.

THESIS 1.1: [C2, C3] *I have given a constructive proof that no datagram network,*

[1] According to [ICM+02], about 50% of the failures spontaneously recover in a minute.

which forwards packets based on destination address and incoming interface, can provide over arbitrary network topology and arbitrary additive edge cost such a local protection, which does not change the header of the packet in any way, and which can guarantee that the following two claims both hold true at the same time:

- *the default path is the shortest one between any two nodes and*
- *detours are always loop-free, even when multiple failures exist.*

Remark: *Since U-turn alternates, FIR and FIFR are IPFRR techniques, which use interface-based forwarding, and shortest paths as default, they can create loops.*

The possibility of loops in detours when interface-based IP fast reroute is used stems from the phenomenon of losing the state of packets. Since the failure of forwarding a packet is indicated by only the incoming interface, if a packet on a detour can get back to a default path, the failure can be "forgotten". Thus, when the forwarding of a packet fails again it will be forwarded along a detour, possibly back to the the original failure. Moreover, if the default paths are the shortest ones between the nodes, safely leaving the detours is not always possible.

Consider the network depicted in Figure 2, suppose that there is FIR in this network and the lengths of links are uniformly 1. Now, if node s wants to send some packets to d, the default path is $s \to f \to e \to d$. However, if link $\{e, d\}$ fails, FIR can immediately restore the connectivity by locally rerouting packets. Since there is no other possibility, e would send the packets back to f. Since interface-based forwarding is used, f would not send the packets back to e, instead it would give them to s, from where they would get to d on path $s \to a \to b \to c \to d$. If there is another failure, say, if link $\{c, d\}$ is down too, packets would reach c as previously. Since c would get packets from b, and since c is the default next hop of b to destination d, there is no way for c to detect that it is the second failure (recall, that only the destination address and the incoming interface are taken into consideration). Thus, c would "think" that this was the first failure, and it would try to bypass the failure, it would send packets back on path $c \to b \to a \to s \to f \to e$, which would create a loop.

Supposing FIFR, the situation is almost the same. Now, consider the network depicted in Figure 3. As it can be observed, now two nodes, node c and e, are failed. In this case f tries to reroute, packets get to b, where they are rerouted again, and in this way a loop is formed.

Furthermore, multiple failures are not always necessary for creating loops; FIR, and the version of FIFR, capable to handle both link and node failures [WN07], can create loops in the case of a single node failure. Consider the network depicted in Figure 4, and suppose that all the link lengths are uniformly 1 and node s sends some packets to d. Now, the shortest path is $s \to f \to c \to d$. If f loses the connectivity

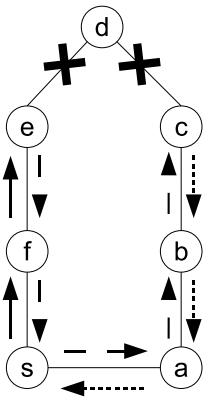


Figure 2. Loop if the link between node d node e and node c node d are both down

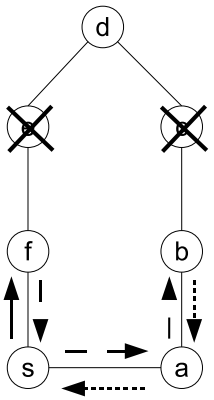


Figure 3. Loop if both node c and e are down

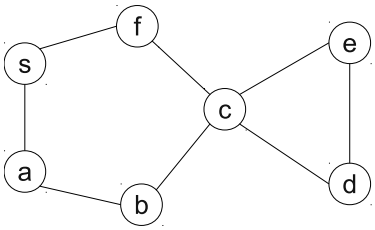


Figure 4. Example network for FIFR loops

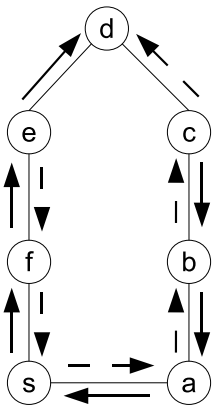


Figure 5. Two edge-disjoint directed spanning trees for node d as a destination

with c, it cannot decide whether the link or the node went down. Since supposing the node failure would mean that there would be no path to d, even FIFR will try to bypass the failure. However, if after all c is the failed resource, packets get to a loop again.

Although, it has been shown that there is no possibility to always avoid loops, when shortest paths are used as default, this also means that there is a possibility, if we give up this requirement.

THESIS 1.2: [C2, C3, P1] *I have proposed a new interface-based IP fast reroute method, called Loop-Free Failure Insensitive Routing (LFIR), which always avoids loops at the price of using non-shortest paths as default paths.*

Remark: *Although I have proposed an IPFRR technique, IP is not necessary. For any datagram network using interface-based forwarding, it is trivial to modify LFIR to be applicable.*

It was discussed previously that the main reason of forwarding loops in detours stems from the possibility of getting back to the default path after a failure. It is possible to avoid this by finding a routing, where packets once entered to detour can never leave it. Observe that in IP networks routing to a given destination is usually done along a directed spanning tree. If it was possible to find a pair of directed edge-disjoint spanning trees for any given destination, packets would be able to follow one of them as default path and the other one as detour.

LFIR follows this idea. Since finding two edge-disjont directed spanning trees rooted at any given node of a 2-edge-connected graph is possible, LFIR can always find distinct default paths and detours in each 2-edge-connected component of the network. In this way, LFIR can handle any single link failure, which does not split the network into two. Moreover, if the forwarding of a packet on detour fails, LFIR drops the packet, and restoration is started immediately. Since packets can only leave detour when they leave a 2-edge-connected component, no loops can be formed. Two directed spanning trees of the former network with node d as destination are depicted in Figure 5.

Now, suppose that again node s sends some packets to d, and the tree marked by solid arrows is the default path. If there is no failure, packets get to d on path $s \to f \to e \to d$. If link $\{d, e\}$ fails, node e redirects the packets to the secondary tree, and packets get to d on path $e \to f \to s \to a \to b \to c \to d$. However, if link $\{c, d\}$ fails too, node c gets a packet on the secondary tree. Since that packet cannot be forwarded along the same tree, the second failure can be detected, packets can be dropped, and restoration can immediately be started. Observe the drawback of this technique: if not s but e.g., a was the source, packets would be forwarded along path $a \to s \to f \to e \to d$ in the intact network. Naturally, this is a suboptimal path.

So a trade-off was found: loops can always be avoided for the price of using longer paths by default. At this point, the most important questions are the probability

Top.	Prob. of loops w/ removed edges	Prob. of loops w/ removed node
NSF	0.39 %	5.4 %
Germany	0.82 %	18.68 %
Italy	0.76 %	18.38 %

Table 1. Average probability of loops when two edges or one node is removed from networks with real topology

Node number	Neighbor	Prob. of loops w/ removed edges	Prob. of loops w/ removed node
20	2	0.32 %	11.64 %
20	3	0.02 %	1.35 %
30	2	0.48 %	22.28 %
30	3	0.05 %	4.95 %
40	2	0.61 %	30.37 %
40	3	0.11 %	9.16 %
50	2	0.7 %	36.4 %
50	3	0.14 %	13.35 %

Table 2. Average probability of loops when two edges or one node is removed from networks generated by BRITE

of creating loops with the original technique, and the lengths of default paths, when LFIR is used. The next two theses answers these questions.

THESIS 1.3: [C2, C3] *By means of extensive simulations on networks with real-world and artificial topologies, I have shown that in the presence of a single node failure or multiple link failures FIR can indeed create loops, in contrast to LFIR, which guarantees loop-free failure recovery.*

For the simulations, I used the topology of both real and randomly generated networks. The topologies of random networks were generated by Boston university Representative Internet Topology gEnerator (BRITE) [MLMB05]. Using these topologies, I found, that it has a significant chance that randomly selected failures can create loops (see Table 1 and Table 2). Observe that this chance is not negligible, it can be even more than 36% for single node failures. Thus, using LFIR undoubtedly has important advantages, but applying this technique is only worth if its drawback, the increased default path lengths are tolerable. The next thesis deals with this problem.

THESIS 1.4: [C2, C3] *I have proposed heuristics in order to reduce the lengths of*

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

Table 3. Average path lengths using LFIR with and without heuristics related to using shortest paths in networks with real topology

Node number	Neighbour	LFIR w/ heur.	LFIR w/o heur.
20	2	105.53 %	128.59 %
20	3	101.68 %	127.61 %
30	2	105.26 %	129.27 %
30	3	101.63 %	129.68 %
40	2	105.04 %	129.65 %
40	3	101.56 %	131.04 %
50	2	104.86 %	129.86 %
50	3	101.5 %	132 %

Table 4. Average path lengths using LFIR with and without heuristics related to using shortest paths in networks with random topology

default paths of LFIR. I have conducted extensive simulations and in the cases I have examined, I have found that these heuristics are effective, and the default paths of LFIR with these heuristics are at most 17% longer in average than the shortest ones.

For the simulations, I used the same configuration as previously, but now I was dealing with the lengths of paths instead of the probability of loops. As one may observe (Table 3 and Table 4), default paths of LFIR are only slightly longer than the shortest ones, at most by 17% in average, when length decreasing heuristics are applied. Moreover, these results also prove, that my heuristics are effective, since they reduced the average length of default paths by 15%–30% of the shortest paths.

4.2 Finding Vertex-Redundant Trees in Linear Time

Although the first group of theses define the first interface-based protection mechanism, which does not suffer from loops, there is a more important experience LFIR has given: appropriate directed trees can be well applied for IPFRR. Unfortunately, the edge-disjoint spanning trees of LFIR can only cover single link failures. Since vertex-disjoint spanning trees are a nonsense, this type of trees cannot be easily improved to

bypass node failures as well. In order to solve the problem, consider Definition 4.1.

Definition 4.1. Let an undirected graph be G with vertex r. A pair of *edge-/vertex-redundant trees* of graph G rooted at r is a pair of directed spanning trees rooted at r, such that there are two paths along the two trees from any given vertex $s \neq r$ to r, and they are edge-/vertex-disjoint respectively.

Remark: Obviously, edge-/vertex-redundant trees can only be found in 2-edge-/2-vertex-connected graphs (these are not only necessary but sufficient criteria), a requirement real networks may not be able to fulfil. Lifting this artificial requirement is addressed by the next group of theses. For this group of theses I always assume that the graph is 2-vertex-connected.

It is easy to prove that edge-disjoint spanning trees and edge-redundant trees are the same. However, the definition of edge-redundant trees gives the possibility of generalizing this concept to vertex-redundant trees, which can provide connectivity even when a node goes down.

Redundant trees [MBFG99, XCT02c, XCT02b, XCT02a, XCT03, ZXTT05, ZXTT08, C5, C6, C7], also known as independent trees [IR84, Han98, ABS96, ZI89, Huc94, MTSIN98, CLY03, CLY06] and colored trees [Ram04, TRK06, BR06, RHK06, RKK07, JRY09, KRKH09], are a well studied objects in graph theory. They were first applied by Itai and Rodeh in [IR84]. They proposed algorithm for finding both edge- and vertex-redundant trees with computational complexity linear in the number of edges. Later, Médard *et. al.* [MBFG99] introduced these graphs as redundant trees to the field of protection in optical networks.

Redundant trees can easily be used for resilience. If a single failure shows up, it can ruin the connectivity towards the root in only one of the trees. Hence, any failure can be bypassed by switching the traffic to the tree which still provides connectivity. In the sequel $V(G)$ denotes the set of vertices and $E(G)$ denotes the set of edges of graph G. A pair of redundant trees is depicted in Figure 6. Since the theses of this group are dealing with vertex-redundant trees, the simple term of "redundant trees" denotes vertex-redundant trees in the sequel.

The first algorithm, which used heuristics for finding a pair of redundant trees in linear time with respect to the overall cost of trees, is called ReducedCostV, and it was presented in [ZXTT05, ZXTT08]. Here, cost means the number of edges of the undirected graph used by *either* of the trees (so the edge counts one, if one or both of the trees use it). However, as it will turn out, this algorithm is not always linear. Therefore, I have proposed a *centralized* algorithm, which is able to handle this problem. Furthermore, in this thesis group, I introduce a *distributed* algorithm as well, which is able to compute not only a single pair of redundant trees, but one routed at each node still in linear time.

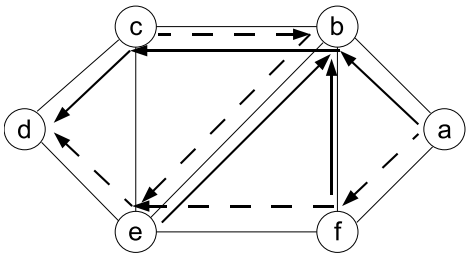


Figure 6. A pair of vertex-redundant trees rooted at vertex d.

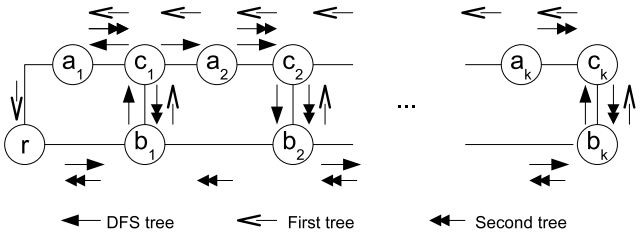


Figure 7. The graph and the DFS traversal, which makes ReducedCostV to fail to find a correct pair of redundant trees in linear time

THESIS GROUP 2: [C5, C6, C7] *I have found a graph and a possible Depth First Search (DFS) traversal, which makes algorithm ReducedCostV to fail to find a correct pair of redundant trees in linear time. I have proposed a centralized algorithm for finding a pair of redundant trees in strictly linear time. Moreover, in order to minimize the overall costs of the trees, my algorithm uses heuristics with quality comparable to the quality provided by the original ReducedCostV. Furthermore, I have proposed a distributed, linear time algorithm for finding a pair of redundant trees rooted at each vertex.*

THESIS 2.1: [C7] *I have found a graph and a possible Depth First Search (DFS) traversal, which makes algorithm ReducedCostV to fail to find a correct pair of redundant trees in linear time.*

Remark: *ReducedCostV can be modified in order to guarantee correctness, but in this way it necessarily loses linear complexity.*

ReducedCostV is based on a decomposition, called "ear-decomposition", of the graph. This decomposition is started from a sole vertex, which will be the root of the redundant trees. First, the root is the only vertex in the trees, then other vertices and edges are added to them. In each step a path or a cycle (called ear) is found in the original undirected graph, and its vertices and edges are added to the trees. However, recall that directed trees are being built, so the direction of the edges does matter; the edges are added in one direction to the first tree, and in the other direction to the second tree. For each ear, it is essential to decide which direction is selected for which tree. In order to answer this question, ReducedCostV keeps up an order of vertices, which is called "voltage".

The problem of ReducedCostV stems from the need of maintaining this order: guaranteeing linear time execution would need data structure with insertion and comparison both in $O(1)$ time. Naturally, usual data structures (e.g., arrays, linked lists, binary trees) cannot fulfil this criterion.

According to private mailing, the authors used platform native numbers. Unfortunately, by associating numbers to vertices an "address" space is created (these are not real addresses, they are used only for comparing), which can run low without proper assignment. Using the graph and the DFS traversal depicted in Figure 7, I have shown that no proper assignment exists; the necessary size of address space scales exponentially with the number of vertices of the graph, which cannot be handled by the fixed size of platform native numbers. I have shown that for 32 bit integers, we only need at most 103 vertices for the algorithm to fail, and for 64 bit integers can run low for graphs containing not more than 199 vertices. Floating point arithmetic does not come to rescue here either: double precision floats of 64 bits run short again at 199 vertices at the most. Arbitrary precision arithmetic can be applied, but it does not provide $O(1)$ comparison or assignment.

Since the problem of ReducedCostV steams from the fact that voltages are used, we can avoid them by completely avoiding voltages. Recall that voltages are needed for comparing the endpoints of ears, so my algorithm always "knows", which endpoint is the lower one, and comparison is not needed anymore.

THESIS 2.2: [C7] *By allowing the algorithm to walk along the Depth First Search (DFS) tree not only downwards but also upwards, I have made an algorithm for finding vertex-redundant trees in any 2-connected graph with strictly linear, $O(|E(G)|)$ complexity. I have conducted extensive simulations and I have found that the cost of the redundant trees found this way is only slightly higher than the ones found by ReducedCostV in the cases I have examined.*

ReducedCostV uses DFS traversal for finding ears; it walks down from some given vertex along the DFS tree till it gets to a proper successor. As it was proven, the path to this successor is a good ear-candidate. The idea of finding proper ears without voltages is based on the observation that it is possible to find an ear not only by walking *down*, but also by walking *up* along the DFS tree. Thus, if we have the vertex with the lowest voltage, which can be an endpoint of an ear, leave this vertex only when *all* its ears are found; for finding all the ears it is needed sometimes to walk up along the DFS tree. These principles are summarized in Algorithm 1.

Moreover, my algorithm uses heuristics in order to decrease the overall cost of the trees. As Zhang *et. al.* have proven [ZXTT05, ZXTT08], the longer ears (paths) found results the lower cost. Therefore, when my algorithm walks down on the DFS tree, it does exactly the same as the original algorithm. Furthermore, when it needs to walk upwards, it maximizes the number of neighbours covered by a sole ear. By using extensive simulations, I have proven that these heuristics are effective, and they can provide redundant trees with slightly higher costs than the ones have, which were found by ReducedCostV.

For simulations, I used again the topology of real and randomly generated networks. As it can be observed (Table 5 and Table 6), my heuristics are effective, since

Algorithm 1 The schematic pseudo code of finding redundant trees rooted at node r in linear time

1: Compute a DFS tree rooted in node r. Let S be an empty stack. Push r into S.
2: **while** S is not empty
3: $x \leftarrow$ pop S
4: Find all the uncovered ears with x as an endpoint (walk both up and down)
5: Add ears to the trees, and keep in mind that x is the lowest vertex
6: Push the vertices of the ears into S in reverse order
7: **end while**

Topology	Number of nodes	Original cost	Cost w/ revised algorithm	Increase
Germany	17	19.82	19.82	0%
NSF	26	31.65	31.8	0.47%
Cost266	37	43.49	44.13	1.47%
Germany50	50	57.06	57.78	1.26%

Table 5. Average cost of redundant trees on real topologies

Node number	Neighbor	Original cost	Cost w/ revised algorithm	Increment
20	2	25.74	26.31	2.21%
20	3	23.94	24.38	1.84%
30	2	38.76	39.73	2.5%
30	3	36.26	37.07	2.23%
40	2	51.76	53.04	2.47%
40	3	48.57	49.69	2.31%
50	2	64.68	66.3	2.5%
50	3	60.76	62.16	2.3%

Table 6. Average cost of redundant trees on topologies generated by BRITE

the cost of redundant trees computed in linear time are less than 2.5% greater for all the topologies studied than the ones computed by the original algorithm.

Vertex-redundant trees can provide vertex-disjoint paths towards the root, so they can be well applied for protecting traffic to a sole destination. Since in real world, communication networks' traffic can flow between any two nodes, it is usually needed to find a pair of redundant trees rooted at not only one, but each vertex. Thus, although finding a pair of redundant trees rooted at a given vertex has $O(|E(G)|)$ complexity, computing a pair of redundant trees with each vertex as a root is not linear any more, it needs $O(|V(G)||E(G)|)$ time. Moreover, since each spanning tree has $|V(G)|-1$ edges, $2|V(G)|$ spanning trees (a pair of them rooted at each vertex) have $2|V(G)|(|V(G)|-1)$ edges. Since only writing these edges into the memory needs at least $2|V(G)|(|V(G)|-1)$ steps, finding all these trees needs $\Omega(|V(G)|^2)$ time, so no centralized algorithm for computing all the trees can be linear.

However, one may also observe that in datagram networks, like IP networks, usually no node (router) needs complete redundant trees, but only the next hops along these trees, the edges emanating from the vertex representing the node. Naturally, in this way all redundant trees are still computed, however the information is now *distributed* in the network; although no node knows the whole trees, the network together can forward packets along them. Now, $|V(G)|$ processors can be used (these are typically the routers in the network), which gives the possibility of further decreasing the computation complexity.

THESIS 2.3: [C5, C6] *I have proposed a distributed algorithm, which can compute a pair of redundant trees rooted at each vertex of a 2-connected graph in linear, $O(|E(G)|)$ time, if there are at least $|V(G)|$ processors, one assigned to each vertex.*

Corollary: *In a network, where each router has computation capability, all the routers can compute the next hops along all the trees in linear time.*

The way of computing a pair of redundant trees rooted at each vertex is based on a special intermediate graph representation called spanning Almost Directed Acyclic Graph (ADAG). The name comes from the fact that these graphs "almost" fulfil the Directed Acyclic Graph (DAG) property; there is a special vertex r, called root, and by removing r, the remaining graph is a DAG. My algorithm supposes that all the processors compute exactly the same spanning ADAG, which can be easily realized, if all of them have the same graph and the same root as input.

The spanning ADAG can be used for defining a partial order. Based on this order a processor assigned to vertex x (which is typically a node in the network) can compute the vertices strictly greater and less than x. Using only this information simple rules can be defined, and computing the outgoing edges of redundant trees rooted at any given vertex can be realized in $O(1)$ time. Thus, computing all the edges takes $O(|V(G)|)$ time, and since computing a spanning ADAG takes $O(|E(G)|)$ time, the overall complexity is $O(|E(G)|)$ as well (the graph is connected), so the

algorithm is linear. These principles are summarized in Algorithm 2.

Algorithm 2 The schematic pseudo code of algorithm finding the edges of redundant trees emanating from u.

1: Find an ADAG
2: Create partial ordering from the ADAG; let V_u^- and V_u^+ be the set of vertices strictly less and greater than u.
3: **for** each vertex d
4: Using some simple rules, compute the edges emanating u belonging to the trees rooted at d.
5: **end for**

4.3 Improving vertex-redundant trees

Previously, some ways to compute redundant trees were shown. In this section, I focus on the trees found. I improve the previous results in two ways.

First, as it was already mentioned (remark of Definition 4.1), redundant trees can only be found in graphs, which are 2-vertex-connected. Since real networks cannot always fulfil this criterion even when they are intact (e.g., see Abeline, AT&T in [SND] or Italian backbone in [GO05]), I have generalized the concept of redundant trees from 2-vertex-connected graphs to arbitrary connected graphs. I named these generalized redundant trees as maximally redundant trees (Definition 4.2). A pair of maximally redundant trees is depicted in Figure 8.

Definition 4.2. Let an undirected graph be G with some vertex r. A pair of *maximally redundant trees* of graph G rooted at r is a pair of directed spanning trees rooted at r, such that there are two paths along the two trees from any given vertex $s \neq r$ to r, and they have the minimum number of vertices in common.

Remark: Having only the cut-vertices in common involves having the minimum number of common edges as well, since an edge having cut-vertices as both endpoints is a cut-edge (bridge).

Maximally redundant trees give us the maximum possible redundancy. If there are two vertex-disjoint paths between two vertices, in maximally redundant trees the paths are vertex-disjoint as well. Otherwise, both paths will only include the minimum number of common vertices, the cut-vertices, which must be contained by any path. Maximally redundant trees can be found in arbitrary connected graphs.

Second, the paths in several real networks are optimized somehow, thus usually it is not enough to simply find maximally redundant trees, but they need to fulfil some extra requirements. As an example, one may consider algorithm ReducedCostV

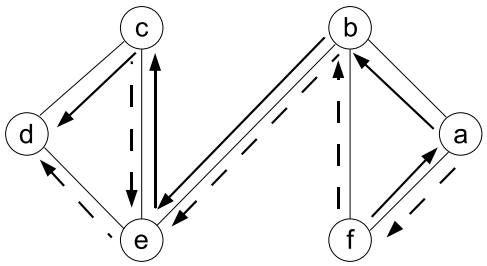


Figure 8. A pair of maximally redundant trees rooted at vertex d.

proposed by Zhang *et. al.* in [ZXTT05, ZXTT08]. As it was discussed in the previous section, this algorithm uses heuristics in order to decrease the overall cost of the found trees.

ReducedCostV was proposed for networks, where exclusively reserving resources (links) for protection is needed. Since in IP networks link costs are selected in such a way that the shortest paths are the ones, which should be chosen, in these networks it is more desirable to select trees, which contain short paths. Therefore, in this section I propose heuristics, which make paths significantly shorter on average while still retaining linear complexity.

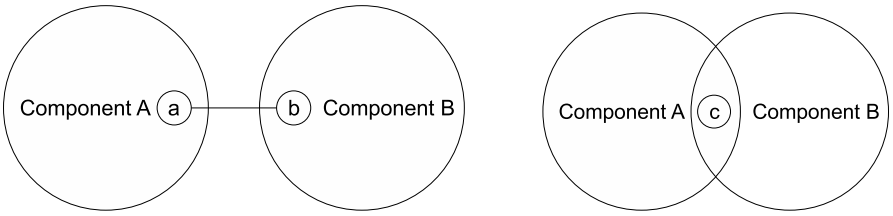
THESIS GROUP 3: [C7, J4] *I have proven that my linear time algorithm for finding redundant trees can find maximally redundant trees as well with only slight modifications. Additionally, I have proven that the distributed algorithm I proposed can also be applied to arbitrary connected graphs with minor modifications, and it finds maximally redundant trees. Furthermore, I have decreased the length of paths in the maximally redundant trees without increasing the computational complexity.*

THESIS 3.1: [C7] *I have proven that the algorithm I proposed for finding redundant trees in linear time (in Thesis 2.2) can be applicable on arbitrary connected graphs with slight modifications. On these graphs my algorithm finds maximally redundant trees and its complexity remains linear, $O(|E(G)|)$.*

The main idea stems from the fact that all graphs are made up by some 2-vertex-connected components (possibly a component may contain only one vertex) and in these components the algorithm is correct and complete. Thus the algorithm is correct and complete, if and only if it can handle the boundaries of these 2-vertex-connected components. The two possible ways of connecting two components are depicted in Figure 9.

Two 2-vertex-connected components can have no or one common vertex, since two components with more than one vertex in common would qualify as a sole component. I have modified my algorithms to handle both of these situations properly.

Furthermore, the same observation leads to the possibility of improving the linear



(a) no common vertex (b) one common vertex

Figure 9. Relations of 2-vertex-connected components

time distributed algorithm, and enable to find the a pair of maximally redundant trees rooted at each vertex.

THESIS 3.2: [J4] *I have improved the distributed algorithm of Thesis 2.3 for finding a pair of maximally redundant trees rooted at each vertex in arbitrary connected graphs in linear, $O(|E(G)|)$ time.*

Using the same observation as previously, namely that connected graphs are made up by some 2-vertex-connected components, the intermediate graph representation can be generalized. This generalized intermediate graph is called Generalized ADAG (GADAG). Using this graph a special partial order of the vertices can be provided again.

This order helps to compute the edges of maximally redundant trees rooted at vertices in the same 2-vertex-connected component in a similar way as it is done for vertex-redundant trees. However, for other trees rooted at the remaining vertices, selecting an appropriate cut-vertex is needed as well, which can be realized by a special recursive algorithm. A schematic pseudo code is presented in Algorithm 3.

Algorithm 3 The schematic pseudo code of algorithm finding the edges of maximally redundant trees emanating from u.

1: Compute the edges of trees rooted at vertices in the same 2-vertex-connected component using Algorithm 2
2: Set the edges towards the global root of the GADAG
3: Using a special "recursive lookup", find the edges belonging to the remaining trees

As it was mentioned previously, in this section I introduce theses, which improve redundant trees. One way of improving is generalizing them to arbitrary connected graphs. However, there are other needs, which are important for applicability in real networks. Therefore, I have proposed some simple heuristics, which decrease the lengths (the number of vertices) of paths along the maximally redundant trees found.

THESIS 3.3: [J4] *I have defined heuristics to reduce the number of vertices along*

Network	Node number	Prim. path w/o heur.	Sec. path w/o heur.	Prim. path w/ heur.	Sec. path w/ heur.
Abilene	12	210%	212%	168%	171%
Germany	17	231%	230%	191%	190%
AT&T	22	221%	224%	166%	167%
NSF	26	224%	222%	178%	174%
Italy	33	248%	247%	175%	174%
Cost266	37	250%	253%	190%	194%
Germany50	50	304%	309%	212%	214%

Table 7. Average number of vertices on paths of maximally redundant trees in real word networks (100% is the path with minimum number of vertices).

the paths in maximally redundant trees, which retain the linear complexity of the distributed maximally redundant tree algorithm. I have conducted extensive simulations on real and random network topologies and I have found that the proposed heuristics reduce the number of vertices along the paths in the distributed maximally redundant trees by 20%–50% in the cases I have examined.

One may observe that the length of maximally redundant trees depends on the GADAG found. One may also observe that usually there are some edges of the graph not used in the GADAG. If it were possible to add these edges to the GADAG, it would decrease the number of hops along the paths in maximally redundant trees. The problem is that these edges cannot be added to a GADAG in an arbitrary direction, as GADAG property must be kept up.

Fortunately, this property can be kept up easily. Thanks to the special spanning GADAG finding technique applied by my algorithm, the found GADAG is such that it can be easily transformed to a DAG D. As I have proven, adding an edge to the GADAG in such a direction, which would not ruin the DAG property of D, is always safe. Therefore, I make a topological ordering on D, and using this order the safe direction can be selected for each of the unused edges in $O(1)$ time. Since a topological ordering takes at most $O(|E(G)|)$ time for a connected graph, finding (maximally) redundant trees remains linear.

Although these heuristics may even increase the length of some paths, when multiple redundant trees are computed by the distributed algorithm, I have shown by extensive simulations that they are effective in the networks I have studied. The results are presented in Table 7 and Table 8.

Node number	Neighbors	Prim. path w/o heur.	Sec. path w/o heur.	Prim. path w/ heur.	Sec. path w/ heur.
20	2	217%	224%	173%	174%
20	3	298%	313%	180%	181%
30	2	235%	243%	182%	182%
30	3	332%	352%	190%	190%
40	2	250%	259%	190%	189%
40	3	361%	385%	198%	197%
50	2	263%	273%	197%	195%
50	3	388%	415%	205%	203%

Table 8. Average number of vertices on paths of maximally redundant trees in artificial networks (100% is the path with minimum number of vertices).

4.4 Lightweight Not-Via

To our days, numerous IP fast reroute solutions came into existence, yet the largest industrial backing among those, which can cover 100% of single failures, is undoubtedly behind the technique based on the notion of "Not-Via addresses" [BSP10]. Although this technique has some drawbacks, Not-Via is still a rather straight-to-the point solution. This thesis group shows the disadvantages of Not-Via and a possible way of eliminating or at least to mitigating them.

The most important drawback of Not-Via is that it uses numerous IP addresses to identify the failed resource, in this way posing substantial address management burden on network operators. Furthermore, Not-Via needs numerous shortest path tree computations, which generates significant additional complexity. Finally, Not-Via needs handling some special cases, making the algorithm hard to trace for debugging or development purposes.

THESIS GROUP 4: [C5, C6, J4, P2] *I have shown that by utilizing the concept of maximally redundant trees for backup paths, the computational and management complexity of Not-via can be reduced to a level that is comparable to the complexity of traditional shortest path routing.*

THESIS 4.1: [C5, C6, J4, P2] *I have redefined the backup path computing method of Not-via, so that detours are organized over maximally redundant trees, and I named this new IPFRR method Lightweight Not-via. I have shown that Lightweight Not-via uses only three IP addresses per node at most, and it asymptotically eliminates the additional complexity penalty that stems from the need to compute backup paths for IP Fast ReRoute.*

Remark: *Lightweight Not-Via does not need additional IP addresses at all in several current IP networks.*

Lightweight Not-via is based on the idea that maximally redundant trees could be used as detours instead of computing numerous shortest paths for backup paths. In an intact network, packets are forwarded along the shortest paths. Moreover, when Lightweight Not-via detects a failure, it does almost the same as Not-via does: it puts the packets into an IP-in-IP tunnel, and tries to send them to the next-next hop, where they are decapsulated, and where they can be forwarded from along the shortest path safely. However, detours are now computed in a completely different way. Now, we compute a pair of maximally redundant trees rooted at each node instead of numerous shortest paths. Note that similar approach was presented in [CHA07], however there redundant trees were applied, which can be found only 2-node-connected networks.

Since a pair of maximally redundant trees provides connectivity after any single failure, which do not split the network into two, the next-next hop must remain reachable along at least one of trees rooted in it. Therefore, packets are encapsulated into a tunnel with IP address describing that the packets should be forwarded along the primary maximally redundant tree. However, if forwarding fails again, this destination address is changed, and packets are now forwarded along the secondary tree. If even forwarding along the secondary tree fails too, packets must be dropped, and restoration must be started immediately, since there are multiple failures in the network.

Observe that using this concept, each node needs at most three IP addresses: one IP address describes the default shortest path, and two describe the maximally redundant trees. Moreover in some IP networks, where each interface has its own address, it is possible to completely avoid using extra addresses: since every node, which can be reached on multiple paths must have at least two interfaces, the addresses of these interfaces can describe the maximally redundant trees, and the loopback address of the node can describe that packet should be forwarded along the shortest path.

Observe that since detours are maximally redundant trees, and a pair of maximally redundant tees rooted at each node can be computed in linear time (Thesis 3.2), computing the paths of Lightweight Not-via is dominated by computing the default paths by a run of Dijkstra's algorithm.

THESIS 4.2: [C5, C6] *I have conducted extensive measurements on an IP Fast ReRoute prototype with real traffic traces and in the cases I have examined, I have found that the extra management and computational burden of the original Not-via is higher than that of the Lightweight Not-via by at least one order of magnitude.*

I used the full-fledged Not-via and Lightweight Not-via prototype network developed and deployed by MSc students I supervised. I have modified these prototypes in order to make them capable to use simulated topologies instead the one explored by OSPF. Then, I measured the computational complexity and the number of addresses needed.

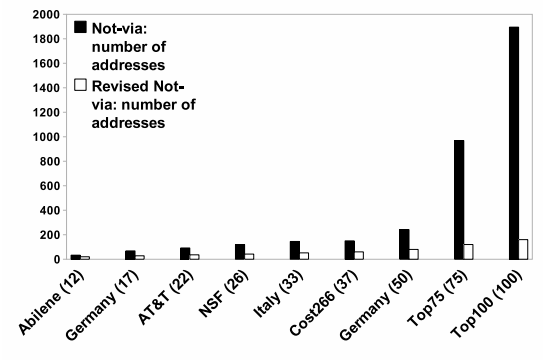


Figure 10. Number of additional addresses for the original and lightweight Not-via in commonplace ISP topologies (number of nodes is given in parentheses), with every fifth node substituted by a LAN.

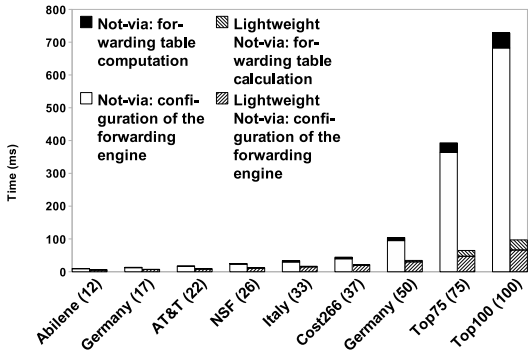


Figure 11. Execution time of computing the forwarding tables and configuring the forwarding engine for the original and the Lightweight Not-via, with every fifth node substituted by a LAN.

Although the behavior of Not-via can be problematic even in point-to-point networks, it becomes completely unmanageable, when Local Area Networks (LANs) are presented. Therefore, I used real and random network topologies with LANs, injected them to the testbed, and measured the IP addresses and the computational complexity needed by both Not-via and Lightweight Not-via. My results are presented in Figure 10 and Figure 11.

As it can be observed, just the sheer number of addresses can prevent us using Not-via. Naturally, setting some thousands of IP addresses by hand is impossible, but it is also problematic with using some centralized management tool. Furthermore, the high number of IP addresses does not manifest itself only in management complexity. Using IP addresses at this rate significantly bounds the routing configuration process too. As it can be observed in Figure 11, the time needed for downloading the freshly computed information into the routing table scales poorly, when Not-via is applied.

In contrast, observe that Lightweight Not-via was able to keep the number of IP addresses low. Moreover, using less IP addresses brought much lower computational time, which in this way reached a level, which is in pair with the time usually needed for current routers for configuration.

5 Application of New Results

Nowadays, there is an increasing interest in IPFRR techniques. Thanks to the convergence of communication networks, currently IP is used more and more often to

transport not only elastic, but also real-time traffic. Since this kind of traffic has different needs, several problems have risen.

One of the unavoidable problems is the need of fast convergence. Although IP is traditionally very robust against failures, the speed of the convergence was never considered as an essential problem. However, low recovery speed is not acceptable for real-time traffic, which will make IPFRR techniques indispensable in some years.

Although the need is increasing, enabling fast recovery in IP networks brings serious problems, and there have been no satisfying proposal. My LFIR algorithm, proposed in Thesis Group 1, eliminates one of the most important problems of IPFRR algorithms using interface-based forwarding, namely that they are prone to form loops. Moreover, my Lightweight Not-via (described in Thesis Group 4) gives possibility to overcome the most important drawback, the extreme management cost, of Not-via. Thus, these IPFRR mechanisms could immediately be deployed to routers, as it has been already done by developing a Lightweight Not-via testbed.

My results, presented in Thesis Group 2 and Thesis Group 3, improve the concept of redundant trees in various ways. These results are essential, since they give the theoretical base needed for Lightweight Not-via. However, applicability of these results is not limited to IPFRR, as redundant trees can be applied in various other fields of telecommunication, e.g., in large sized multiprocessor systems [IR84], in circuit switched networks [MBFG99, XCT02c, XCT02b, XCT02a, XCT03, ZXTT05, ZXTT08], or even in sensor networks [Ram04, TRK06, BR06, RHK06, RKK07, JRY09].

6 Summary of works related to theses

6.1 Paper [C2]: A Novel Loop-Free IP Fast Reroute Algorithm

G. Enyedi, G. Rétvári, T. Cinkler
13th EUNICE Open European Summer School;la BEST PAPER AWARD
Paper is related to Thesis 1.1, 1.2, 1.3, 1.4.

Summary: This paper describes algorithm LFIR. It shows that there can be loops, when the original FIR is used and that LFIR use not significantly more resource for forwarding.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out simulation experiments, interpreted the results, and wrote the article under the supervision of the second and the third author.

6.2 Paper [C3]: A Loop-Free Interface-Based Fast Reroute Technique

G. Enyedi, G. Rétvári
4th EURO-NGI Conf. on Next Generation Internet Networks (EuroNGI)
This paper is related to Thesis 1.1, 1.2, 1.3, 1.4.

Summary: This paper is an improved version of the previous one. Describes LFIR as well, but there are additional details and more simulation results.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out simulation experiments, interpreted the results, and wrote the article under the supervision of the second author.

6.3 Paper [C5]: IP Fast ReRoute: Lightweight Not-Via without Additional Addresses

G. Enyedi, P. Szilágyi, G. Rétvári, A. Császár
IEEE INFOCOM-MiniConference
This paper is related to Thesis 2.3, 4.1, 4.2.

Summary: This mini-conference paper that briefly shows the problems of Not-via, supports these claims by measurement results, describes Lightweight Not-via for 2-connected networks (so there only redundant trees are applied) and introduce the

distributed algorithm for finding multiple redundant trees in linear time.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out measurements experiments, interpreted the results, and wrote a significant part of the article under the supervision of the third and fourth author. The testbed was developed by the second author. The remaining part of the paper was written by the third author.

6.4 Paper [C6]: IP Fast ReRoute: Lightweight Not-Via

G. Enyedi, G. Rétvári, P. Szilágyi, A. Császár
IFIP Networking
This paper is related to Thesis 2.3, 4.1, 4.2.

Summary: This is the full paper version of the previous one. The problems of Not-via and technique Lightweight Not-via are presented with additional details.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out measurements experiments, interpreted the results, and wrote a significant part of the article under the supervision of the second and fourth author. The testbed was developed by the third author. The remaining part of the paper was written by the second author.

6.5 Paper [C7]: On Finding Maximally Redundant Trees in Strictly Linear Time

G. Enyedi, G. Rétvári, A. Császár
IEEE Symposium on Computers and Communications, ISCC
This paper is related to Thesis 2.1, 2.2, 3.1.

Summary: In this paper, we show that the algorithm of Zhang may fail to find a proper pair of redundant trees in linear time. We give the improved, centralized algorithm, which is now strictly linear. Finally, the way of improving this algorithm for finding maximally redundant trees is also discussed.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out simulation experiments, interpreted the results, and wrote the article under the supervision of the second and the third author.

6.6 Paper [J4]: Finding Multiple Maximally Redundant Trees in Linear Time

G. Enyedi, G. Rétvári
Periodica Polytechnica Electrical Engineering
This paper is related to Thesis 3.2, 3.3, 4.1.

Summary: In this paper, we describe the distributed algorithm for finding maximally redundant trees. There the heuristics decreasing the lengths along these trees are also presented. Finally, we show how maximally redundant trees can be applied with Lightweight Not-via, in this way enabling it to be used in arbitrary networks.

Contribution: The first author of this paper formulated the problem, developed the mathematical model, carried out simulation experiments, interpreted the results, and wrote the article under the supervision of the second author.

6.7 Patent application [P1]: Link failure recovery method and apparatus

A. Császár, G. Enyedi
Patent application
This paper is related to Thesis 1.2.

Summary: Patent application related to technique LFIR. Technique LFIR is published in [C2, C3].

Contribution: The second author of this paper formulated the problem, developed the mathematical model, and wrote the patent application together with the first author.

6.8 Patent application [P2]: Lightweight Not-Via IP Fast Reroute

A. Császár, G. Enyedi
Patent application
This paper is related to Thesis 4.1.

Summary: Patent application related to technique Lightweight Not-via. Technique Lightweight Not-via is published in [C5, C6, J4].

Contribution: The second author of this paper formulated the problem, developed the mathematical model, and wrote the patent application together with the first author.

References

[ABS96] F. Annexstein, K. Berman, and R. Swaminathan. Independent spanning trees with small stretch factors. Technical report, 1996.

[Atl06] A. Atlas. U-turn alternates for ip/ldp fast-reroute. Internet Draft, available online: `http://tools.ietf.org/html/draft-atlas-ip-local-protect-uturn-03`, February 2006.

[BR06] R. Balasubramanian and S. Ramasubramanian. Minimizing average path cost in colored trees for disjoint multipath routing. In *15th International Conference on Computer Communications and Networks, ICCCN 2006*, pages 185–190, October 2006.

[BSP10] S. Bryant, M. Shand, and S. Previdi. IP fast reroute using Not-via addresses. Internet Draft, available online: `http://tools.ietf.org/html/draft-ietf-rtgwg-ipfrr-notvia-addresses-06`, 2010.

[CHA07] T. Cicic, A. F. Hansen, and O. K. Apeland. Redundant trees for fast IP recovery. In *Broadnets*, pages 152–159, 2007.

[CLY03] S. Curran, O. Lee, and X. Yu. Chain decompositions and independent trees in 4-connected graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 186–191, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[CLY06] S. Curran, O. Lee, and X. Yu. Finding four independent trees. *SIAM Journal on Computing*, 35(5):1023–1058, 2006.

[fS02] International Organization for Standardization. OSI IS-IS intra-domain routing protocol. ISO/IEC 10589:2002, 2002.

[GO05] M. L. Garcia-Osma. TID scenarios for advanced resilience. Tech. Rep., The NOBEL Project, Work Package 2, Activity A.2.1, Advanced Resilience Study Group, Sep 2005.

[Han98] Dagmar Handke. Independent tree spanners. In *Graph-Theoretic Concepts in Computer Science*, pages 203–214, 1998.

[Huc94] A. Huck. Independent trees in graphs. *Graphs and Combinatorics*, 10(1):29–45, 1994.

[ICM+02] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 237–242, New York, NY, USA, 2002. ACM.

[IR84] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. In *SFCS '84: Proceedings of the 25th Annual Symposium onFoundations of Computer Science, 1984*, pages 137–147, Washington, DC, USA, 1984. IEEE Computer Society.

[JRY09] G. Jayavelu, S. Ramasubramanian, and O. Younis. Maintaining colored trees for disjoint multipath routing under node failures. *IEEE/ACM Transactions on Networking*, 17(1):346–359, 2009.

[KRKH09] S. Kini, S Ramasubramanian, A. Kvalbein, and A. F. Hansen. Fast recovery from dual link failures in IP networks. INFOCOM 2009, April 2009.

[LYN+04] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. In *IEEE Infocom'04*, 2004.

[MBFG99] M. Médard, R. A. Barry, S. G. Finn, and R. G. Galler. Redundant trees for preplanned recovery in arbitary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking*, 7(5):641–652, Oct 1999.

[MLMB05] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Boston university Representative Internet Topology gEnerator. `http://www.cs.bu.edu/brite`, 2005.

[Moy98] J. Moy. OSPF version 2. Internet Engineering Task Force: RFC 2328, April 1998.

[MTSIN98] K. Miura, D. Takahashi, S.-I.Nakano, and T. Nishizeki. A linear-time algorithm to find four independent spanning trees in four-connected planar graphs. In *WG '98: Proceedings of the 24th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 310–323, London, UK, 1998. Springer-Verlag.

[NLY+07] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Transaction on Networking*, 15(2):359–372, 2007.

[NLYZ03] S. Nelakuditi, S. Lee, Y. Yu, and Z.-L. Zhang. Failure insensitive routing for ensuring service availability. In *Proceedings International Workshopon Quality of Service (IWQoS)*, 2003.

[Ram04] S. Ramasubramanian. Supporting multiple protection strategies in optical networks. Technical report, Department of Electrical and Computer Engineering, University of Arizona, November 2004.

[RHK06] S. Ramasubramanian, M. Harkara, and M. Krunz. Distributed linear time construction of colored trees for disjoint multipath routing. In *IFIP Networking*, pages 1026–1038, May 2006.

[RKK07] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz. Disjoint multipath routing using colored trees. *Computer Networks*, 51(8):2163–2180, 2007.

[SB10] M. Shand and S. Bryant. IP Fast Reroute framework. Internet Engineering Task Force: RFC 5714, January 2010.

[SND] Survivable fixed telecommunication Network Design library (SNDlib). `http://sndlib.zib.de`.

[TRK06] P. Thulasiraman, S. Ramasubramanian, and M. Krunz. Disjoint multipath routing in dual homing networks using colored trees. In *IEEE Globecom*, pages 1–5, November/December 2006.

[WN07] J. Wand and S. Nelakuditi. IP fast reroute with failure inferencing. In *Proc. of ACM SIGCOMM Workshop on Internet Network Management – The Five-Nines Workshop*, 2007.

[WZN06] J. Wang, Z. Zhong, and S. Nelakuditi. Handling multiple network failures through interface specific forwarding. In *IEEE Globecom*, November 2006.

[XCT02a] G. Xue, L. Chen, and K. Thulasiraman. Cost minimization in redundant trees for protection in vertex-redundant or edge-redundant graphs. In *PCC '02: Proceedings of the Performance, Computing, and Communications Conference, 2002. on 21st IEEE International*, pages 187–194, Washington, DC, USA, 2002. IEEE Computer Society.

[XCT02b] G. Xue, L. Chen, and K. Thulasiraman. Delay reduction in redundant trees for preplanned protection against single link/node failure in 2-connected graphs. In *IEEE Globecom*, November 2002.

[XCT02c] G. Xue, L. Chen, and K. Thulasiraman. QoS issues in redundant trees for protection in vertex-redundant or edge-redundant graphs. In *IEEE International Conference on Communications (ICC)*, volume 5, pages 2766–2770, 2002.

[XCT03] G. Xue, L. Chen, and K. Thulasiraman. Quality-of-service and quality-of-protection issues in preplanned recovery schemes using redundant trees. *IEEE Journal on Selected Areas in Communications*, 21(8):1332–1345, October 2003.

[ZI89] A. Zehavi and A. Itai. Three tree-paths. *Journal of Graph Theory*, 13(2):175–188, 1989.

[ZNY+05] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C-N Chuah. Failure inferencing based fast rerouting for handling transient link and node failures. In *IEEE Infocom'05*, 2005.

[ZXTT05] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman. Linear time construction of redundant trees for recovery schemes enhancing QoP and QoS. INFOCOM 2005, March 2005.

[ZXTT08] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman. Faster algorithms for construction of recovery trees enhancing QoP and QoS. *IEEE/ACM Trans on Networking*, 16(3):642–655, 2008.

Publication of new results

[J] Journals

[J1] András Császár, **Gábor Enyedi**, Gábor Rétvári, Marcus Hidell, Peter Sjödin, "Converging the Evolution of Router Architectures and IP Networks", *IEEE Network Magazine*, Special Issue on Advances in Network Systems Architecture, 21:4(8–14) 2007.

[J2] Péter Fodor, **Gábor Enyedi**, Gábor Rétvári, Tibor Cinkler, "Layer-Preference Policies in Multi-layer GMPLS Networks", *Photonic Network Communications* 2009.

[J3] **Gábor Enyedi**, Gábor Rétvári, "Gyors hibajavítás IP hálózatokban (Hungarian)", *Híradástechnika*, 64:3–4(20–24) 2009.

[J4] **Gábor Enyedi**, Gábor Rétvári, "Finding Multiple Maximally Redundant Trees in Linear Time", *Accepted to Periodica Polytechnica Electrical Engineering*, available online: `http://opti.tmit.bme.hu/~enyedi/ipfrr/`, 2010.

[C] Conferences

[C1] Péter Fodor, **Gábor Enyedi**, Tibor Cinkler, "A Fast and Efficient Traffic Engineering Method for Transport Networks", V Workshop in G/MPLS Networks (WGN5), pages 129–141, 2006.

[C2] **Gábor Enyedi**, Gábor Rétvári, Tibor Cinkler, "A Novel Loop-free IP Fast Reroute Algorithm", 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services (EUNICE), Twente, The Netherlands, 2007. BEST PAPER AWARD

[C3] **Gábor Enyedi**, Gábor Rétvári "A Loop-Free Interface-Based Fast Reroute Technique", 4th EURO-NGI Conf. on Next Generation Internet Networks (EuroNGI), Krakków, Poland, pages 39–44, 2008.

[C4] Péter Fodor, **Gábor Enyedi**, Gábor Rétvári, Tibor Cinkler, "An Efficient and Practical Layer-preference Policy for Routing in GMPLS Networks", 13th Int. Telecommunications Network Strategy and Planning Symposium (NETWORKS), Budapest, Hungary, 2008.

[C5] **Gábor Enyedi**, Péter Szilágyi, Gábor Rétvári, András Császár, "IP Fast ReRoute: Lightweight Not-Via without Additional Addresses", IEEE INFOCOM-MiniConference, Rio de Janeiro, Brazil, April 2009.

[C6] **Gábor Enyedi**, Gábor Rétvári, Péter Szilágyi, András Császár, "IP Fast ReRoute: Lightweight Not-Via", IFIP Networking, Aachen, Germany, May 2009.

[C7] **Gábor Enyedi**, Gábor Rétvári, András Császár, "On Finding Maximally Redundant Trees in Strictly Linear Time", IEEE Symposium on Computers and Communications, ISCC, Sousse, Tunisia, July 2009.

[C8] Gábor Rétvári, János Tapolcai, **Gábor Enyedi**, András Császár "IP Fast ReRoute: Loop-free Alternates Revisited", Accepted to IEEE INFOCOM, available online: `http://opti.tmit.bme.hu/~enyedi/ipfrr/`, Shanghai, China, April 2011.

[P] **Patent Applications**

[P1] András Császár, **Gábor Enyedi**, "Link failure recovery method and apparatus", Patent application WO/2009/010090, 2009.

[P2] András Császár, **Gábor Enyedi**, "A System and Method of Implementing Lightweight Not-via IP Fast ReRoutes in a Telecommunications Network", Patent application WO/2010/055408, 2010.

[P3] András Császár, **Gábor Enyedi**, "Fast Path Notification", Patent application PCT/EP2010/059391, 2010.

[P4] András Császár, **Gábor Enyedi**, "IP Fast ReRoute Relying on Fast Path Notification", Patent application PCT/EP2010/065040, 2010.