IP Fast ReRoute: Lightweight Not-Via

Gábor Enyedi[1], Gábor Rétvári[1,⋆], Péter Szilágyi[1], and András Császár[2]

[1] Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Magyar tudósok körútja 2., Budapest, Hungary, H-1117
{enyedi,retvari,szilagyi}@tmit.bme.hu
[2] TrafficLab, Ericsson Research
Laborc utca 1., Budapest, Hungary, H-1037
Andras.Csaszar@ericsson.com

Abstract. In order for IP to become a full-fledged carrier-grade transport technology, a native IP failure-recovery scheme is necessary that can correct failures in the order of milliseconds. IP Fast ReRoute (IPFRR) intends to fill this gap, providing fast, local and proactive handling of failures right in the IP layer. Building on experiences and extensive measurement results collected with a prototype implementation of the prevailing IPFRR technique, Not-via, in this paper we identify high address management burden and computational complexity as the major causes of why commercial IPFRR deployment still lags behind, and we present a lightweight Not-via scheme, which, according to our measurements, improves these issues.

Keywords: QoS, resilience, IP, fast reroute, redundant trees.

1 Introduction

IP has come a long way to become a cost-effective bearing platform for commercial services, providing scalable QoS, point-and-click management, secure VPN services, unpaired scalability, etc. There is, however, an important piece still missing in the puzzle: a resilience scheme capable to treat transient and persistent failures in some tens of milliseconds. Nowadays, IP networks rely on the somewhat outdated resilience scheme built into routing protocols, like Open Shortest Path First (OSPF), hardly fast enough for multimedia applications. Hence, operators resort to working around the limitations of IP, deploying for instance MultiProtocol Label Switching (MPLS) Fast ReRoute, and tolerate the implied boost in capital and operational expenditures.

In response to these challenges, the Internet Engineering Task Force has initiated the IP Fast Reroute framework [1] to introduce fast, local and proactive failure recovery in IP networks. "Local", in this context, means that only routers in the vicinity of the failed component repair, and "proactive" means that detours are precomputed well in advance. To our days, many IPFRR proposals have

⋆ G. Rétvári was supported by the János Bolyai Fellowship of the Hungarian Academy of Sciences.

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 157–168, 2009.

come to existence, yet the largest industrial backing is undoubtedly behind the technique based on the notion of "Not-via addresses" [2]. In order to distinguish detoured packets from ordinary packets, so that special routing can be applied to them, Not-via introduces an alternative address space: packets tunneled around the failed component are destined to certain not-via addresses, clearly separable from normal addresses and unambiguously communicating which component the sender believes to be the cause of the failure. This way, detours simplify into shortest paths in a topology with the failed component deleted, and rerouting boils down to pushing the packet to a pre-established IP-in-IP tunnel, a cheap operation now commonly implemented in the fast-path of IP routers. This makes Not-via a practical and easy-to-deploy IPFRR technique.

This paper came into being in reaction to the vast operational experience we gathered on a Not-via-enabled IP testbed deployed at BME-TMIT [3]. Thanks to our prototype system, we are now in a position to be able to thoroughly judge on Not-via's pros and cons. We found that Not-via raises serious address management issues, originating from the need to hand out many not-via addresses, and it poses substantial additional CPU-load on IP routers. This is objectionable, as contemporary IP infrastructure, even without IPFRR, is actually struggling to keep up with the ever-increasing routing tables. The significant additional management and computational cost makes operators reluctant to adopt IPFRR, despite of its potential benefits.

To improve the manageability of Not-via, we present a lightweight Not-via scheme. The main idea is, on the traces of [4], to adopt the concept of node-redundant trees (simply redundant trees in the sequel) for IPFRR and apply them directly to Not-via. As shall be shown, this modification reduces the number of not-via addresses, cuts the computational complexity down to the level of plain shortest path routing, and it removes many corner cases that plague the original Not-via proposal.

The rest of the paper is organized as follows. In Section 2, we discuss Not-via and we summarize our operational experiences. In Section 3, we recast Not-via over redundant trees, we discuss the issue of additional addresses and we report on a related theoretical result: a distributed algorithm which finds next-hops in redundant trees corresponding to all nodes in linear time. In contrast, this was only possible in quadratic time or worse previously. We implemented the modified Not-via in our prototype and in Section 4 we present observations and measurement results we gathered on our testbed. Finally, in Section 5 we conclude the paper.

2 IPFRR Using Not-Via Addresses

IP Fast ReRoute attains fast response time by handling failures locally, with only the routers in the vicinity of the failure participating in the repair but other, distant routers not being informed of the failure in any ways. Therefore, IPFRR applies special routing to packets being forwarded along a detour. Otherwise, loops might emerge as a distant router not aware of the failure might blindly loop the detoured packet back along the default forwarding path. Not-via uses the

destination address in IP packets to mark whether the packet is being forwarded on the default path or in an IP-to-IP tunnel along a detour. The starting node of the detour is the router whose next-hop has become unreachable, and the tunnel is terminated at the next-next-hop (NNH), the second closest node along the shortest path tree. This facilitates common handling of node and link failures.

Perhaps an example is in order here. Consider the network depicted in Fig. 1, and suppose that a packet entering the network at node A is forwarded to the egress node C. Furthermore, assume that the shortest path (marked by bold arrows) goes through node B, but A suddenly loses contact with B. Now, B encapsulates the packet in a new IP header with a special not-via address set as destination address, which has the semantics "forward me to C (the NNH) not via B", and sends it to node D. Thanks to the special destination address, D will not send the packet back to C on the shortest path (as it would be the case if default routing applied), but instead sends it to E through LAN L. Node E forwards the packet to node C where it is decapsulated and passed further along the default forwarding path as if no failure happened.

Unfortunately, Not-via is a bit more difficult than that, and anyone trying to implement it faces painful exceptions and complex corner cases. Consider, for instance, the so called *LAN problem* that arises when D tries to send a packet to node E using LAN L but LAN L fails. On one hand, node D could assume that all the nodes connected to this LAN failed, in which case it would lose all connectivity to node E. On the other hand, if selective fault detection was available on the LAN, then D could distinguish between a LAN failure (when more than one router attached to the LAN becomes unavailable) and multiple single router failures. This would provide more efficient recovery, at the cost of quadratic number of additional not-via addresses to cover all the possible fault scenarios. Similar corner cases arise at the decapsulation point of the detours (the so called *last-hop problem*) and at bridge nodes [2].

Despite these issues, Not-via is still a practical and rather straight-to-the-point solution, therefore, we chose Not-via to base our IPFRR testbed onto. After dealing with all the intricacies of implementing the standard and experimenting with it in operation, we are now feeling confident enough to judge on Not-via's merits and identify some of its pressing limitations.

Burdening address management: The first question an implementor inevitably faces is how to assign and distribute not-via addresses. As of this writing, there is no official protocol support for advertising not-via addresses into the routing domain. The situation is worsened by the fact that a not-via address has a compound meaning, as it encodes both a destination node and a component to be avoided, and there is currently no way to communicate this rich semantics between routers. As a work-around, network operators may assign not-via addresses statically, but this is inflexible, subject to human configuration errors and breaks down rapidly as the network increases. Just the sheer number of not-via addresses can pose problems: the simple network of Fig. 1 would require a total of 17 not-via addresses.

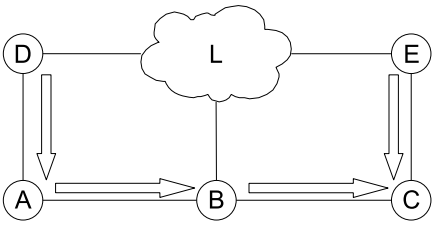


Fig. 1. Sample network with IP routers A, B, C, D and E and LAN L. Bold arrows mark the shortest path to C.

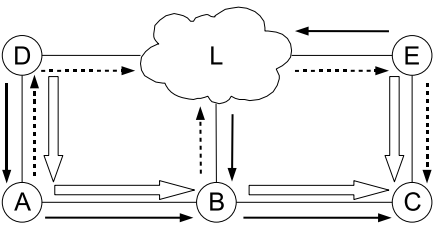


Fig. 2. Sample network. Bold arrows mark the shortest path to C, dashed arrows mark the primary and solid arrows mark the secondary redundant tree rooted at C

Considerable computational overhead: In an ordinary IP network, the next-hops towards all destinations are obtained by a single shortest path tree (SPT) calculation. With Not-via, a router must execute as many SPT instances as there are components that can fail, with the failed component removed from the topology. Using some simple heuristics one can go down to some few dozen additional SPT calculations [5], which is still significant. Note that substantial additional costs show up due to having to deal with an increased number of entries in the routing tables, establish, maintain and tear down tunnels, etc.

Complexity and special cases: As mentioned above, Not-via brings in subtle intricacies into routing and in many cases it overrides well-known IP routing mechanisms. The corner cases mentioned above make implementations convoluted and operation of the protocol hardly tractable by operators.

In Section 4, we shall support the above claims with measurement results obtained on an operational IP testbed. In addition, we note that similar observations were reported elsewhere [5]. In the next section, we propose deliberate modifications to Not-via in order to remove, or at least mitigate, these compelling issues.

3 An Improved Lightweight Not-Via

Our modified Not-via technique uses the concept of redundant trees [6]. Redundant trees are basically a pair of directed spanning trees, which have the appealing property that a single node or link failure destroys connectivity through only one of the trees, leaving the path along the other tree intact. The concept was first applied to IP Fast ReRoute in [4]. In contrast, in this paper we apply redundant trees directly to the prevailing IPFRR technique, Not-via. As shall be shown below, organizing the detours over redundant trees gives rise to an easily implementable and deployable "lightweight Not-via" scheme: it significantly decreases the number of Not-via addresses, with clever modifications it reduces computational complexity to linear, and it eliminates most of Not-via's corner cases without introducing new ones.

3.1 Redefining the Semantics of Not-Via Addresses

Our lightweight Not-via uses ordinary shortest paths for default forwarding, and a pair of redundant trees (the *primary* and the *secondary* backup tree) for resilience. Correspondingly, a node v has three IP addresses: a default ($\mathcal{D}_v$), a primary ($\mathcal{P}_v$) and a secondary ($\mathcal{S}_v$). If there is no failure, packets are forwarded along the shortest paths as usual. On the other hand, if a failure shows up, packets are tunneled along either the primary or the secondary tree. This is achieved by encapsulating the packets into a new IP header with the primary (respectively, secondary) address of the Next-next-hop set as the outer destination address. Since, by definition of redundant trees, a single failure leaves at least one of the trees intact, it is guaranteed that packets avoid the failed component.

Consider Fig. 2, depicting the same sample network as before, but now not only the shortest path but also the primary and the secondary backup trees directed towards node C are given (observe that the paths in these trees are node-disjoint). Suppose A has a packet to send to node C. As long as its default next-hop, B, is alive, A simply passes the packet to B. If, however, B goes down, A must find a backup path, or at least a next-hop that can push the packet further, towards C. So it encapsulates the packet, sets the outer destination address to the primary backup address of the NNH (node C) and passes it to the next-hop along the primary tree, node D. Assuming that D computed the exact same redundant tree to C (which is not hard to ensure), D will pass the packet through LAN L and node E to C, where it gets decapsulated and sent further. If, instead, it is now node E that has to get a packet to C and it finds that connectivity to C went away, both its shortest path and its primary backup path are affected by the failure. In this case, the packet is encapsulated to the secondary backup path and sent through L to B. Note that the secondary backup path can not be impacted by the failure in this case, as it is node disjoint from the primary path. Finally, a packet forwarded along the primary path gets rerouted to the secondary path should it encounter a failure on its path (this might be the very same failure that pushed the packet to the backup in the first place) but not *vice versa*.

The forwarding process of the lightweight Not-via scheme is given in Algorithm 1. Note that the operation `push` $\mathcal{X}$ in routing terminology means "encapsulate the packet into an IP-in-IP tunnel and set its outer destination address to $\mathcal{X}$". The operation $\mathcal{X} \leftarrow$ `pop` does the reverse: decapsulates the packet and puts the address of the innermost IP header to $\mathcal{X}$. $\mathcal{D}$, $\mathcal{P}$ and $\mathcal{S}$ are the address spaces of the default, primary and secondary backup addresses.

It is easy to see intuitively that this forwarding rule is correct. First, in the absence of failures, packets get to their destination along the shortest path as usual. In case of a single failure, a packet first gets to the NNH along either the primary or the secondary backup path, provided that such paths exist, which is always true as long as the network is 2-connected (see more on this matter later). Both backups can not be affected by the failure at the same time, as they are node disjoint. So single node or link failures are handled correctly. Finally, packets can not get into loops in the presence of multiple simultaneous failures,

Algorithm 1. Forwarding process at node u for a packet destined to address $\mathcal{A}$, given the set of unavailable neighbors F. The next hop of $\mathcal{X}$ is given by $\text{nh}(\mathcal{X})$.

```
1: if A = P_u or A = S_u then                          # This is the end of the tunnel
2:     A ← pop;
3: end if
4: if A = D_u then                                     # This is the destination
5:     consume the packet; return ;
6: end if
7: if nh(A) ∉ F then                                   # Next hop is operational
8:     forward packet to nh(A); return ;
9: end if
10: if A ∈ D then                                      # Default path failed
11:     let v be the NNH to A;
12:     if nh(P_v) ∉ F then                            # Forward to primary next hop
13:         push P_v and forward packet to nh(P_v); return ;
14:     else if nh(S_v) ∉ F then                       # Forward to secondary next hop
15:         push S_v and forward packet to nh(S_v); return ;
16:     end if
17: else if A ∈ P then                                 # Primary backup path failed
18:     X ← pop;
19:     S_X ← the secondary backup address for X;
20:     if nh(S_X) ∉ F then                            # Forward to secondary next hop
21:         push S_X and forward packet to nh(S_X); return ;
22:     end if
23: end if
24: drop the packet;                                   # Secondary backup failed
```

as a packet is unconditionally dropped should it encounter a failure along the secondary path.

With this modification, a not-via address protects multiple failures; the primary address protects the default path and the secondary address protects the primary backup. In this way, the number of addresses is decreased to 3 per node, the absolute minimum realizable by the original Not-via only in special topologies (point-to-point rings). What is more, in certain cases it is possible to completely avoiding using extra addresses. In traditional IP networks, routers have a loopback address and a unique address for each interface. Hence, one can use any two interface addresses as the primary and the secondary address. Since these addresses are always disseminated by the IGP, other routers can easily learn them. Naturally, in this case routers should be addressed via their loopback, otherwise traffic destined directly to routers would not be protected. While in a conventional IP network this technique removes the need to maintain additional not-via addresses, it must be emphasized that it is not applicable to any arbitrary IP network. Namely, IP backbones running over unnumbered point-to-point links (e.g., MPLS LSPs) still need to maintain at least two additional addresses per router, since interfaces usually don't have unique IP addresses in such cases.

We have seen previously that the original Not-via proposal has some subtle details, making it difficult to implement it correctly and understand it in operation. Though, redefining Not-via in terms of redundant trees removes most of the corner cases. For instance, LANs no more need special treatment: a LAN is handled like any ordinary node except that it does not get not-via addresses. Additionally, in [7] we show an easy way to tackle the problem of bridge nodes that show up in non-2-connected networks, another corner case in Not-via. Finally, the last-hop problem is treated by simply repairing to the next-hop, similarly to Not-via (as a matter of fact, we have already seen this case when we examined the case of E sending a packet to C and losing connectivity to it).

3.2 Reducing the Computational Complexity of Not-Via

The computational cost of Not-via is dominated by the large number of SPT calculations, since an SPT with respect to all potentially failing components needs to be obtained. Suppose there are N nodes and E point-to-point links in a network and there are no LANs. In this case, Not-via's complexity is $O(N(N \log N+E))$ (N times the complexity of Dijkstra's SPT algorithm), which is worse than quadratic in the number of nodes. Unfortunately, without careful modifications the lightweight Not-via would have essentially the same complexity: although a pair of redundant trees comes in linear time, $O(E)$ [8], we need redundant trees with respect to all destination nodes yielding $O(NE)$ steps in general. In this section, we show how to reduce this complexity to $O(E)$ using a simple distributed algorithm.

The idea is that for our lightweight Not-via to work correctly, we do not need the entire redundant tree instances to all destinations, we just need the corresponding next-hops. Thus, we compute a single pair of redundant trees, the primary P and the secondary S, rooted at some designated node r. Then, for any $d \neq r$ we rewire these trees with d set as root, and we take the corresponding next-hops along the rebased trees. Since computing the initial redundant tree takes $O(E)$ steps and, as shall be shown below, we can decide on the next-hops for a particular node in $O(1)$, the overall complexity is $O(N + E) = O(E)$.

Proposition 1. *Let P and S be a pair of redundant trees, rooted at some r, and perform the following steps to obtain a graph D:*

1. *reverse the edges in S*
2. *take the union of the edges of the resultant trees*
3. *split r into two nodes, r^+ and r^-, so that edges only enter r^+ and only leave r^-*

We assume that P and S were so that the graph D yielded by the above steps is:

(i) a directed acyclic graph (DAG) and
(ii) there is only one edge entering r^+.

While these requirements seem somewhat strong, in reality the majority of the redundant tree algorithms in the literature easily satisfy Proposition 1. We used

the linear time algorithm in [8]. An alternative is to modify a redundant tree algorithm so that it immediately produces the DAG (a good candidate would be the algorithm in [9]). Henceforward, we shall assume that the DAG D is at our disposal, it satisfies *(i)* and *(ii)* and it can be computed in linear time.

Definition 1. *Let the node set of D be V and define a relation $(\prec)$ on V as follows: $u \prec v : u, v \in V$ if and only if there is a directed path from u to v in D.*

It is easy to see that $(V, (\prec))$ makes up a bounded partially ordered set (poset). Because D is a DAG, $(\prec)$ is unambiguous. Additionally, since edges only leave r^-, the minimal element is exactly r^-. Similarly, r^+ is the maximal element.

Definition 2. *For some node u, let V_u^+ be the set of the nodes larger than u. Similarly, let V_u^- be the set of nodes smaller than u:*

$$V_u^+ = \{v \in V | u \prec v\} \ , \qquad V_u^- = \{v \in V | v \prec u\} \ .$$

Additionally, let $f_u^+(d)$ denote the first hop along some path from u to any $d \in V_u^+$, and $f_u^-(d)$ be the same for any $d \in V_u^-$. For the root node r, we define $f_r^+(d) = f_{r^-}^+(d)$ and $f_r^-(d) = f_{r^+}^-(d)$ for all $d \in V \setminus \{r^+, r^-\}$.

V^+ and $f^+(\cdot)$ can be computed by a Breadth-First-Search (BFS) traversal of D. Similarly, V^- and $f^-(\cdot)$ come from a reverse BFS. This way, $f^+(\cdot)$ and $f^-(\cdot)$ encode the next-hop along the minimum-hop path, which makes our detours shorter. Note that in general $V_u^+ \cap V_u^- = \emptyset$ but $V_u^+ \cup V_u^- \neq V \setminus \{u\}$, because some nodes might not be ordered with respect to u.

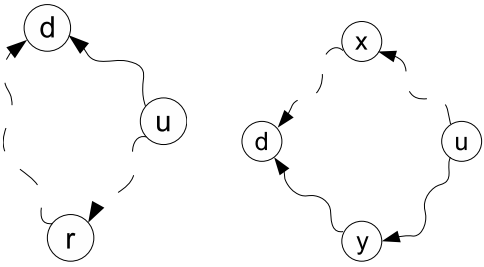
Theorem 1. *Given nodes u and d, $u \neq d$, choose the primary next-hop $h_u^P(d)$ and the secondary next-hop $h_u^S(d)$ from u to d as follows:*

1. *If $d \in V_u^+$: $h_u^P(d) = f_u^+(d)$ and $h_u^S(d) = f_u^-(r^-)$*
2. *If $d \in V_u^-$: $h_u^P(d) = f_u^+(r^+)$ and $h_u^S(d) = f_u^-(d)$*
3. *Else: $h_u^P(d) = f_u^-(r^-)$ and $h_u^S(d) = f_u^+(r^+)$*
4. *Special rules apply at the root node (if $u = r$):*
   $h_r^P(d) = f_r^+(d)$ and $h_r^S(d) = f_r^-(d)$

Then, interleaving the primary next-hops $h^P(d)$ and the secondary next-hops $h^S(d)$ makes up a pair of redundant trees rooted at d.

Proof. To prove the theorem, it is enough to show that following the primary and the secondary next-hops comprises two loop-free, node-disjoint paths. The rules encode the intuitive idea: following the next-hops $h^P(d)$ we move in increasing direction in the poset, along $h^S(d)$ in decreasing direction, and if u and d are not mutually ordered, we move downwards in the poset until we can move upwards (and *vice versa*).

First, we show that for two nodes $v, w : v \prec w$, what we obtain by following the primary next-hops $h^P(w)$ is a loop-free $v \to w$ path. Observe that either $w = f_v^+(w)$ and we arrive to w in the next step, or $w \in V_x^+ : x = f_v^+(w)$ and we can step to $h_x^P(w)$ and repeat the same reasoning to eventually arrive to



(a) ordered case **(b)** unordered case

Fig. 3. Illustration for Theorem 1

w. Along the similar lines, following $h^S(w)$ yields a loop-free $v \to w$ path for $v, w : v \succ w$.

If $d = r$, the claim is trivial. Suppose $d \neq r$ and there is an ordering between u and d, say $u \prec d$. Now, following $h^P(d)$ yields an $u \to d$ path p_p (the path marked by solid arrow in Fig. 3a), and following $h^S(d)$ yields first a $u \to r^-$ path p_s^1 and then an $r^+ \to d$ path p_s^2 (dashed arrow in Fig. 3a). Based on the observation above, these subpaths are indeed paths and they are loop-free. The concatenation of p_s^1 and p_s^2 gives the secondary path p_s. Finally, p_p and p_s are node-disjoint: nodes along p_p belong to the interval $[u, d]$, p_s^1 to $[r^-, u]$ and p_s^2 to $[d, r^+]$, and these intervals are disjunct except the endpoints.

If there is no ordering between u and d, the situation is slightly more difficult: following $h^P(d)$ first yields an $u \to y$ path p_p^1 and then a $y \to d$ path p_p^2, where y is the first node for which $u \succ y$ and $y \prec d$ holds (see the solid arrows in 3b). Similarly, $h^S(d)$ yields first a $u \to x$ path p_s^1 and then an $x \to d$ path p_s^2 for the first $x : u \prec x$ and $x \succ d$ (dashed arrows in 3b). Again, concatenation of the corresponding subpaths yields two node-disjoint paths: first, p_p^1 and p_s^1 are node-disjoint because $p_p^1 \in V_u^-$, $p_s^1 \in V_u^+$ and $V_u^- \cap V_u^+ = \emptyset$; second, p_p^1 and p_s^2 are also node-disjoint because the nodes of p_p^1 are not ordered with respect to d but those of p_s^2 are; third, p_P and p_S can not both traverse r, because $x \prec r^+$ (due to Proposition 1, condition *(ii)*, we have a node m for which $v \prec m : v \in V \setminus \{r^+, m\}$, so the secondary path turns back in m at the very latest). Similar reasoning applies to see that the rest of the subpaths are mutually node-disjoint too. □

Hence, computing the primary and the secondary next-hops with respect to each node in the network involves first obtaining a pair of redundant trees, then converting them to a DAG using Proposition 1, two BFS traversals to compute V^+ and V^- and finally cycling through all nodes to compute the corresponding next-hops using Theorem 1. All these steps can be performed in linear time, therefore, the overall complexity of our method is $O(E)$. Thanks to these modifications, now it takes $O(N \log N + E)$ steps to compute the default next-hops and an extra $O(E)$ steps specific to IPFRR. Therefore, the computational complexity of the lightweight Not-via is dominated by the cost of standard shortest path routing and the additional penalty of IPFRR simply disappears in the long run.

Finally, we point out that basically any protection and restoration scheme relying on redundant trees faces with the problem of finding redundant trees to all destinations at the same time. Therefore, the result that this can be done in linear time might have generic interest beyond IPFRR.

4 Performance Evaluation

In this paper, we argue that it is not some deep theoretical limitation or trade-off that hampers the wide-scale deployment of IPFRR the most, but rather a couple of very technical and very concrete practical issues. In order to confirm this claim, we implemented and tested both the prevailing IPFRR proposal, Not-via, and also our lightweight Not-via in an operational IP testbed. Our test system is a full-fledged IPFRR prototype, deployed on 9 PC routers running a stock Debian GNU/Linux distribution, the Open Shortest Path First routing protocol (OSPF) from the `Quagga` suite of routing daemons [10] and `kbfd`, a kernel-based implementation of the Bidirectional Forwarding Detection [11] protocol[1]. Below, we briefly report on some of our most important observations. For a complete coverage on the measurement results, the reader is referred to [3].

Our experiences indicate that IP Fast ReRoute is just what it promises to be: fast. Configuring BFD so that any failure is detected in at most 9 ms (BFD interval = 3 ms, BFD multiplier = 3), Not-via repairs single failures in 16.65 ms on average and 18.5 ms at maximum. With conventional OSPF, on the other hand, one can measure anything between 120 ms and several seconds depending on the actual topology, the nature and the location of the failure, etc.

Our measurements were primarily aimed at identifying the management cost of Not-via. We found that considerable management complexity arises from the need to hand out and maintain vast numbers of not-via addresses. Fig. 4 gives this number for both the original Not-via and our lightweight Not-via, as computed by our prototype system for some commonplace ISP topologies. To simulate the effect of LANs, we treated 20% of the routers as if they were LANs. Observe that with the lightweight Not-via, the number of additional addresses remains modest even in very large topologies. We found, in addition, that the second most important cost of Not-via comes from its considerable computational complexity. Fig. 5 shows the CPU time needed to compute the default and the backup next-hops and downloading them into the forwarding engine.

These measurement results cast Not-via in a completely different light: although the computational complexity of Not-via is substantial, yet it is the extra management burden caused by the extension of the address pool that dominates its complexity. Our measurements reproduce this burden spectacularly even in small and middle-sized topologies, and we expect it to become prohibitive in larger networks. On the other hand, it is exactly this burden where the advantages of the

[1] Our modifications to `Quagga` and `kbfd` are maintained separately at `http://opt.tmit.bme.hu/~kbfd` until all of our upstream patches go into the respective production releases.

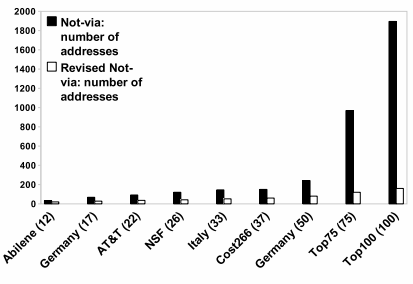


Fig. 4. Number of additional addresses for the original and lightweight Not-via in commonplace ISP topologies (number of nodes is given in parentheses), with every fifth node substituted by a LAN

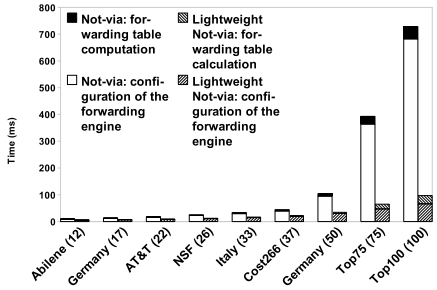


Fig. 5. Execution time of computing the default and backup next-hops and configuring the forwarding engine for the original and the lightweight Not-via

lightweight Not-via really manifest themselves: the time of computing the next-hops and configuring the forwarding engine decreases by an order of magnitude into the range of some few hundred milliseconds, which falls well within the time range contemporary IP routers perform ordinary shortest path routing [12].

5 Conclusion

IP Fast ReRoute is one of the last missing technological components from the IP protocol suite on its way to become a mature carrier-grade transport technology. In this paper, we argued that, despite of the strong incentives, wide-spread adoption of IPFRR will not occur until the additional cost of IPFRR is reduced to a level tolerable to network operators. To support our claims, we presented a formal performance evaluation of IPFRR obtained on a full-fledged prototype. As far as we are aware of, this is the first time that such an evaluation is published in the literature.

Our measurements showed that the immense number of not-via addresses imposes considerable load on both IP routers and network management. However, by reformulating Not-via in terms of redundant trees we could decrease the number of additional addresses substantially. We also improved the complexity of computing the detours to strict linear time from the worse than quadratic complexity of Not-via. Hence, in the lightweight Not-via the extra computational complexity of fast reroute amortizes as compared to even shortest path routing. We discovered, however, that a more significant improvement comes from redefining the semantics of Not-via addresses, so that one address covers not just one but many failure scenarios, since fewer additional addresses caused a spectacular drop in the associated management cost. This demonstrates that, with clever modifications to Not-via, the extra load of IPFRR can be brought down to a tolerable level. We believe that this will further incentivize network operators to seriously consider deploying IPFRR in the future.

References

1. Shand, M., Bryant, S.: IP Fast Reroute framework. Internet Draft (February 2008), http://tools.ietf.org/html/draft-ietf-rtgwg-ipfrr-framework-08
2. Bryant, S., Shand, M., Previdi, S.: IP fast reroute using Not-via addresses. Internet Draft (February 2008), http://www.ietf.org/internet-drafts/draft-ietf-rtgwg-ipfrr-notvia-addresses-00.txt
3. Szilágyi, P., Tóth, Z.: Design, implementation and evaluation of an IP Fast ReRoute prototype. Technical report, BME (2008); First prize at Scientific Student Conference (2008), http://opti.tmit.bme.hu/~enyedi/ipfrr/
4. Cicic, T., Hansen, A.F., Apeland, O.K.: Redundant trees for fast IP recovery. In: Broadnets, pp. 152–159 (2007)
5. Li, A., François, P., Yang, X.: On improving the efficiency and manageability of NotVia. In: Proc. of ACM CoNEXT, pp. 1–12 (2007)
6. Médard, M., Barry, R.A., Finn, S.G., Galler, R.G.: Redundant trees for preplanned recovery in arbitary vertex-redundant or edge-redundant graphs. IEEE/ACM Transactions on Networking 7(5), 641–652 (1999)
7. Enyedi, G., Rétvári, G., Császár, A.: On finding maximally redundant trees in strictly linear time. In: IEEE ICC (2008) (submitted), http://opti.tmit.bme.hu/~enyedi/ipfrr/
8. Zhang, W., Xue, G., Tang, J., Thulasiraman, K.: Linear time construction of redundant trees for recovery schemes enhancing QoP and QoS. In: INFOCOM 2005 (March 2005)
9. Enyedi, G., Rétvári, G.: Finding redundant trees in linear time. IEEE Communications Letters (2008) (submitted), http://opti.tmit.bme.hu/~enyedi/ipfrr/
10. GNU Quagga routing software, http://www.quagga.net
11. Katz, D., Ward, D.: Bidirectional forwarding detection. Internet Draft (2008), http://tools.ietf.org/html/draft-ietf-bfd-base-08
12. Shaikh, A., Greenberg, A.: Experience in black-box OSPF measurement. In: IMW 2001: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pp. 113–125 (2001)