On Improving the Efficiency and Manageability of NotVia

Ang Li[§] Pierre Francois[†] Xiaowei Yang[§*]

[§] Dept. of Computer Science
University of California, Irvine
{angl, xwy}@uci.edu

[†] Dept. of CSE
Université catholique de Louvain
pfr@info.ucl.ac.be

ABSTRACT

This paper presents techniques that improve the efficiency and manageability of an IP Fast Reroute (IPFRR) technology: NotVia. NotVia provides the IPFRR service for all destinations in an ISP's network upon any single link or node failure, while previous proposals such as Loop-free Alternates (LFA) can not guarantee this level of coverage. However, NotVia increases the computational and memory costs of the IPFRR service, and poses new challenges to network management, as routers are unaware of the links and nodes (hence the amount of traffic) that they actually protect. This paper introduces three techniques: NotVia aggregation, prioritized NotVia computation, and the rNotVia algorithm that collectively reduce the overhead of NotVia and improve its manageability. We use simulations to evaluate these techniques on real ISP topologies as well as on randomly generated topologies. The results show that the computational and memory overhead of NotVia are reduced to a fraction of their previous values on various topologies, suggesting that the techniques proposed in this paper make NotVia a more efficient and easy-to-manage IPFRR solution.

1. INTRODUCTION

The trend of digital convergence calls for highly reliable IP networks. Service disruptions as short as a few hundred milliseconds may lead to user-perceivable quality degradation for real-time applications such as IPTV or VoIP [9, 20]. Yet traditional network failure recovery at best takes on the order of a few hundred milliseconds [15, 19, 31].

IP Fast Reroute (IPFRR) is an emerging technology that aims to recover IP forwarding service as soon as a failure is detected using IP-based schemes. IPFRR has significant advantages over traditional network recovery. First, it reduces service disruption time, because a router can locally recover its forwarding service as soon as it detects a failure, instead of relying on distributed routing convergence. Second, it allows routers to suppress transient failures to improve routing stability. Routers can delay the dissemination of failure information in hope that transient failures (e.g. link flappings due to environmental noise) may shortly recover on their own. Third, IPFRR does not require a network to support MPLS. This reduces both the management and upgrade overhead for networks that do not currently use MPLS.

IPFRR has gained much attention from both industry and academia due to its attractive features. Recent discussions at IETF [28] suggest that a combination of Loop-Free Alternates (LFAs) and NotVia addresses be one of the most promising techniques for standardization. This is because the combination can provide protection paths for all destinations upon any single link or node failure in a network. Loop-Free Alternate is a lightweight technique that pre-computes alternate next hops if they exist. A router forwards packets via an LFA next hop if its shortest path next hop fails. However, there is no guarantee that LFAs exist at any router for all destinations. As ISPs desire to offer predictable services, NotVia is introduced to complement LFAs to achieve full failure coverage. If a router does not have an LFA next hop when the link to its shortest path next hop fails, a special address: a NotVia address, is assigned to the shortest path next hop. All routers in the network compute a next hop to reach the NotVia address by removing the failed link. Upon a failure, packets are encapsulated using the NotVia address and sent along a protection path that does not use the failed link to reach the next hop. Node failures are handled in a similar fashion.

[*] A. Li and X. Yang are partially supported by NSF award 0627166. P. Francois is supported by Cisco Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) or originators and do not necessarily reflect the views of Cisco Systems or NSF.

CoNext'07, December 10-13, 2007, New York, NY, USA.
Copyright 2007 ACM 978-1-59593-770-4/07/0012 ...$5.00.

The NotVia technique introduces nontrivial computational and memory overhead. Computing paths for NotVia addresses is time consuming, as each NotVia address requires a shortest path tree (SPT) computation on a different network topology. Once a topology change happens, all routers need to compute as many SPTs as the number of directed links not fully protected by LFA in the network to restore the NotVia state. In addition, each NotVia address increases a router's IGP table size by one. Furthermore, the computations provide no information on which links/nodes a router protects. This situation increases the difficulty for network management, as the network operator cannot predict traffic load on a router or link when a failure occurs. In the case that a router or link cannot handle the overload caused by IPFRR, packets will be dropped and the effort of IPFRR becomes futile.

The goal of this paper is to improve the efficiency and manageability of NotVia. We identify two effective techniques: address aggregation and prioritized NotVia computation to reduce the time required to restore the NotVia state for failure protection and the forwarding table sizes. In addition, we propose a novel algorithm called rNotVia that allows a router to efficiently compute the set of links and routers it protects.

The main contributions of this work are the NotVia aggregation and prioritization techniques that make NotVia more efficient, and the rNotVia algorithm that makes it management-friendly. A secondary contribution is our experimental evaluation on the performance of the LFA and NotVia based IPFRR framework. We evaluate our improvements and rNotVia on real ISP topologies ranging from small to large tier-1 ISP as well as randomly generated topologies using BRITE [26]. We quantify computational costs dedicated to restore the NotVia state as well as the forwarding table size increments. To the best of our knowledge, we are the first to conduct an evaluation on the overhead of NotVia. The results show that without our proposed improvements, NotVia computations may take more than 2 seconds on modern processors (Pentium D, 3.4GHz) and the NotVia address entries may increase the forwarding table sizes of large ISPs by nearly one thousand. With our improvements, both computational costs and forwarding table size increments can be reduced to a fraction of their previous values, especially on large topologies. This suggests that the techniques we propose are likely to work well in practice.

The rest of this paper is organized as follows. § 2 presents background information on the IPFRR framework. § 3 describes the techniques we use to improve NotVia. § 4 specifies the rNotVia algorithm that allows a node to obtain the protection path information. § 5 presents the evaluation. § 6 compares and contrasts our work with related work. § 7 concludes the paper.

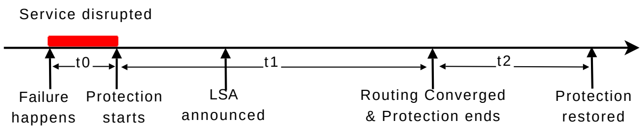


Figure 1: This figure shows the various events triggered by a failure and their time line.

2. BACKGROUND ON IP FAST REROUTE

In this section, we first provide the background information on the current IPFRR framework. We then describe how NotVia works in detail, and analyze its overhead and limitation.

2.1 Overview

The goal of IPFRR is to locally recover IP forwarding service before routing convergence. Figure 1 depicts various events triggered by a failure. The time period t_0 is the failure detection time at a router, and t_1 is the routing convergence time. With traditional network recovery, forwarding service may be disrupted for a time period of $t_0 + t_1$.

An ideal IPFRR solution is to reduce the service disruption time to as short as the failure detection time t_0. To achieve this goal, the IPFRR framework [29] requires routers to pre-compute alternate forwarding paths that do not use a failed link or node. We refer to such paths as protection paths. The IPFRR framework also includes techniques that prevent micro-loops during routing convergence [6,12,14,32]. Therefore, packets can be forwarded along the protection paths without looping during the routing convergence time t_1. Once routing has converged, routers can forward packets along the normal forwarding paths on the new topology, and IPFRR protection ends.

With IPFRR, the network also needs to recompute the protection paths on the new topology after a topology change. This is shown in the time period t_2 in Figure 1. Only after protection paths are restored, the network is ready to protect another failure. Thus, reducing the time spent on computing protection paths is critical in improving the network's ability to provide fast reroute service when failures occur within short intervals.

2.2 The NotVia Fast Reroute Scheme

NotVia [7] is a technique that provides protection paths for any single link or node failure for all destinations in an intra-domain network. NotVia is designed to complement LFA: a NotVia protection path is used when an LFA protection path does not exist. We first briefly describe how a node computes a LFA protection path, and then describe the NotVia protection path.

Let S denote a node that implements the NotVia fast

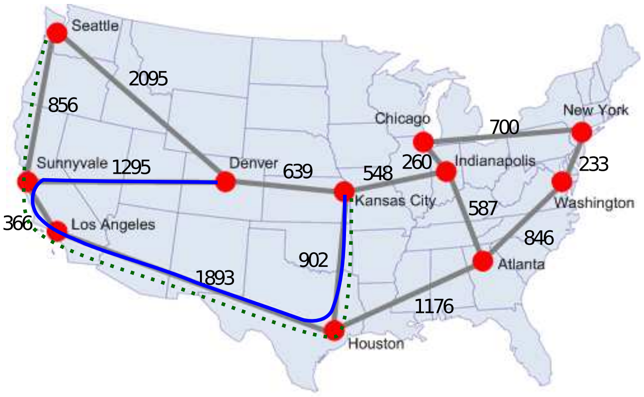


Figure 2: **This figure shows examples of the NotVia protection paths in the Abilene topology. The numbers next to a link are link costs. The solid line shows the protection path from Denver to reach Kansas City when the link to Kansas City fails, and the dotted line shows the protection path from Seattle to Kansas City when the Denver node fails.**

reroute scheme. Suppose S tries to protect its link $S \rightarrow T$ that it uses to reach a destination d. S will first find a Loop-Free Alternate next hop N that does not use the link $S \rightarrow T$ to reach d. Formally, a Loop-Free Alternate node N that protects the link $S \rightarrow T$ for destination d is defined as

$$cost(N \rightarrow \ldots d) < cost(N \rightarrow \ldots S) + cost(S \rightarrow \ldots d)$$

where $cost(X \rightarrow \ldots Y)$ is the shortest path cost from node X to node Y [4]. This condition ensures that packets with destination d rerouted to N will not be forwarded back to S, and will eventually reach the destination d.

For example, in Figure 2, Sunnyvale is a Loop-Free Alternate next hop from Seattle to Indianapolis. If the link to its shortest path next hop (Denver) fails, the Seattle node can forward packets to Sunnyvale.

However, a node S may not have a Loop-Free Alternate neighbor to protect all destinations it reaches via a link $S \rightarrow T$ [13]. For instance, in Figure 2, Denver does not have a Loop-Free Alternate to reach Indianapolis. NotVia protection paths are used in this situation to provide the IPFRR service.

A NotVia protection path is established by a special type of address: the NotVia address. If a node S does not have a Loop-Free Alternate neighbor to protect all destinations it reaches via the link $S \rightarrow T$, a NotVia address $NV_{S \nrightarrow T}$ is assigned to node T and will be used by T's neighbor S. S and all other nodes will compute a protection path to reach the NotVia address $NV_{S \nrightarrow T}$ by removing the directed link from the neighbor S to the node T. We refer to the neighbor node S as the head node of the NotVia address $NV_{S \nrightarrow T}$. When S cannot reach T via the link $S \rightarrow T$, it will encapsulate the packets using the NotVia address $NV_{S \nrightarrow T}$. As all nodes have computed a path to reach this address without using the link $S \rightarrow T$, the packets will finally reach T not via the link $S \rightarrow T$. The packets will then be decapsulated by T and forwarded normally afterwards. Note that unlike LFA, $NV_{S \nrightarrow T}$ is unique to protect link $S \rightarrow T$ regardless of the destination.

Notation	Meaning
$NV_{S \nrightarrow T}$	NotVia address for T to protect link $S \rightarrow T$
$NV_{\cancel{S} \rightarrow T}$	NotVia address for T to protect node S
$cost(S \rightarrow \ldots d)$	Shortest path cost from S to d
pNV_S	NotVia prefix for S in partial aggregation

Table 1: Summary on Notations

In Figure 2, a NotVia address $NV_{Denver \nrightarrow Kansas}$ will be assigned to the Kansas City node for the neighbor Denver as it does not have an LFA next hop to reach Indianapolis. All routers compute a protection path to reach the address $NV_{Denver \nrightarrow Kansas}$ by removing the link from Denver to Kansas City. The protection path from Denver to Kansas City is marked by the solid line in Figure 2. When the Denver node detects a failure to reach Kansas City, it will encapsulate a packet with the address $NV_{Denver \nrightarrow Kansas}$. This packet will be forwarded along the protection path, i.e. Sunnyvale, Los Angeles, Houston, and then Kansas City. At Kansas City, the packet will be decapsulated, and forwarded to its original destination. As can be seen, this packet is successfully rerouted around the failed link between Denver and Kansas City.

NotVia can also be used to protect node failures. Similarly, if when a node S fails, and an upstream node U of the node S (a node that uses S as its next hop) does not have an LFA to reach a destination d, a NotVia address $NV_{\cancel{S} \rightarrow T}$ is assigned to S's next hop T for this d. All nodes compute a path to reach the address $NV_{\cancel{S} \rightarrow T}$ of T by removing the node S. When U detects the failure of S, and S' next hop to reach destination d is T, it encapsulates a packet destined to d with the address $NV_{\cancel{S} \rightarrow T}$. As all nodes compute the next hop to reach this address without using the node S, the packet will reach T not via the node S. Here it is possible to have multiple NotVia addresses to protect the same node failure, given the multiple next hops of the protected node towards different destinations.

We note that when a node protection NotVia address is used, no additional NotVia addresses are required to provide link protection. To reach a link-protection NotVia address $NV_{S \nrightarrow T}$, S can simply use the node-protection NotVia address $NV_{\cancel{S} \rightarrow T}$, and compute a next hop to reach the address $NV_{\cancel{S} \rightarrow T}$ by removing the link $S \rightarrow T$, rather than removing itself. This is correct because at any router on the protection path from S to T (except S), the next hop to reach $NV_{S \nrightarrow T}$ is the same as the next hop to reach $NV_{\cancel{S} \rightarrow T}$. We provide a simple proof for this in Appendix A.

When a node detects that it cannot reach a next hop, it cannot immediately tell whether it is a link failure or a next hop node failure. Therefore, it is recommended [7] that a node U that cannot reach its next hop S should always use a node-protection path to completely bypass S if it can. That is, it should encapsulate its packets with the node NotVia address $NV_{\not{S}\rightarrow T}$ to reach its next-next hop T. A node will use a link-protection path to reach its immediate next hop only when a node-protection path to reach the next-next hop does not exist.

As the same address $NV_{\not{S}\rightarrow T}$ can be used as both a link NotVia address at the node S and a node NotVia address at other nodes, we only use the notation $NV_{\not{S}\rightarrow T}$ to represent a NotVia address in the rest of the paper. It should be interpreted as not via the node S by all nodes other than S, and not via the link $S \rightarrow T$ by the node S.

Figure 2 shows an example of a node-protection NotVia path. If the Denver node fails, the Seattle node can encapsulate its packets destined to the downstream Kansas City with the NotVia address $NV_{Den\not{v}er\rightarrow Kansas}$. The protection path is shown as the dotted line in the figure. Note that the solid line and dotted line overlap with each other from Sunnyvale to Kansas City, as for the nodes on the protection paths, the next hops to reach Kansas City without using the link from Denver to Kansas City are the same as those without using the Denver node.

NotVia addresses can be assigned to protect all nodes from the failure of their neighbors. But only those assigned to protect nodes or links that are not protected by LFA may be computed and actively in use. In this paper, without specific notice, we use the term NotVia addresses to refer to those that might be actively used in a given network topology.

2.3 NotVia Overhead

NotVia introduces non-trivial computational overhead. After a topology change, the network needs to update forwarding entries of normal IP addresses as well as the NotVia address entries. Only after all those entries are updated, the network can fast reroute traffic without causing loops or packet drop when another failure occurs. For each NotVia address, a node must compute a shortest path tree on the network topology without a NotVia node or link. According to analysis on real ISP topologies [13], over 40% links and nodes are not protected by Loop-Free Alternates. For a large Tier-1 ISP hundreds of NotVia addresses are needed for those unprotected links and nodes, and thus hundreds of shortest path tree (SPT) computations are needed. Although each SPT computation can be optimized by using the incremental shortest path first algorithm [17] and early terminating the calculation once the destination is reached, this overhead may still become a problem for large networks. NotVia also increases the forwarding table size by adding additional NotVia address entries. Furthermore, after a topology change, a link or node that is previously protected by a Loop-Free Alternate may not be protected any more and vice versa. A router may need to send out link state announcements to inform others to compute for necessary NotVia addresses or to avoid computing for unnecessary NotVia addresses. This incurs message overhead. If this overhead is high, it may delay the restoration time of NotVia protection paths. We evaluate this overhead in Section 5.

2.4 Difficulty to Predict NotVia Traffic

The current design of NotVia does not provide a router the information on whether it is on a protection path of a NotVia address. Without this information, a router cannot export this information to its Management Information Base [25]. Consequently a network operator has little knowledge on how traffic will be rerouted when a failure occurs. If rerouted traffic passes through a congested link, it may still be dropped, rendering IPFRR futile. On the other hand, if a network operator has the information on what NotVia addresses a router protects, he can adjust OSPF weights [11, 21, 30] or use other traffic engineering techniques [10, 18] to avoid rerouting protected traffic to a congested link. Although the protection information can be provided by some offline management tools such as running protection simulation on real topology, we argue that enabling routers to generate this information has two advantages: (1) routers can update the protection information immediately after the topology change, while offline approaches work only when stable topology information is available; (2) letting routers know the protection information opens room for further features such as automatic protection path adjustment. Thus, it is valuable to provide a router the information on which NotVia addresses it protects.

In the following sections, we describe our techniques that reduce the overhead of NotVia, and an algorithm rNotVia that allows a router to efficiently compute the set of NotVia addresses it protects.

3. IMPROVING THE EFFICIENCY OF NOTVIA

In this section, we describe techniques that reduce the time it takes to recompute NotVia entries. Those techniques can also reduce the number of NotVia address entries in a router's forwarding tables.

3.1 NotVia Aggregation

The first technique is NotVia aggregation. It is based on the intuition that next hops for the NotVia addresses of a node might be the same as the next hops for the regular IP addresses of the node, because they are computed from topologies only differing by one link or node. A node R can efficiently determine whether the next hop to a node T's regular IP address and the next hop to a NotVia address $NV_{\not{S}\rightarrow T}$ of T are the same by examining whether the link $S \rightarrow T$ is on its shortest path to a destination. If it is not, then the two entries at R must have the same next hop. Thus, if the NotVia address $NV_{\not{S}\rightarrow T}$ is assigned from the node T's regular IP address space, it can be aggregated into T's regular entry at R. R does not need additional NotVia computations or forwarding table entries for this NotVia address of T: $NV_{\not{S}\rightarrow T}$, as regular routing convergence will compute a next hop to reach T's regular IP address prefix. This technique also reduces the additional NotVia entry $NV_{\not{S}\rightarrow T}$ in a node's forwarding tables.

In Figure 2, if the NotVia address $NV_{De\not{n}ver\rightarrow Kansas}$ is allocated from the regular IP prefix of the Kansas City node, then the Houston node can aggregate this address into the IP prefix of the Kansas City node, and does not need additional computation to set the next hop for the NotVia address.

NotVia aggregation requires that each router be configured with an IP prefix that covers both its interface addresses and NotVia addresses. Network operator might need to renumber a router to satisfy this requirement. Although we think it can be done in practice, it is possible that network operator prefers to allocate NotVia addresses from a separate NotVia prefix, not to mix them with the regular IP addresses of a router. In this case, a neighbor-specific NotVia address is allocated from this NotVia prefix. All other nodes set the next hop to the NotVia prefix of this node to be the same as the next hop to the regular IP prefix of the node. An entry for a neighbor-specific NotVia address of the node is aggregated into the NotVia prefix entry if they have the same next hop. We refer to this technique as partial aggregation.

In the previous example, suppose the node at Kansas City decides to use a separate prefix pNV_{Kansas} for NotVia addresses. It will allocate all its NotVia addresses ($NV_{De\not{n}ver\rightarrow Kansas}$, $NV_{Indi\not{a}napolis\rightarrow Kansas}$, etc.) under this prefix. Thus Houston node can aggregate the addresses $NV_{De\not{n}ver\rightarrow Kansas}$ and $NV_{Indi\not{a}napolis\rightarrow Kansas}$ into pNV_{Kansas}, and does not need additional computation to set the next hop for pNV_{Kansas}.

With partial aggregation, the number of NotVia computations can still be reduced, as no additional computations for the NotVia prefix or the aggregated NotVia addresses are needed. The reduction of forwarding table sizes is less than that of full aggregation, as each NotVia prefix requires an additional entry.

3.2 Prioritized NotVia Computation

The second technique, prioritized NotVia computation, is designed to shorten the time it takes for the network to restore the NotVia state needed for failure protection after a topology change. The shorter the restoration time is, the sooner a network can recover its protection service, i.e., t_1+t_2 will be shorter in Figure 1.

The prioritization technique makes a node compute NotVia addresses in the order of their "closeness" to it. "Closeness" is defined in terms of hop count. Intuitively, in a well-engineered network, nodes on the protection path to a failure should be close to the failure. We define those NotVia addresses for which a node is on the protection paths as necessary NotVia addresses to the node. To recover the IPFRR service, a node should update the entries for necessary NotVia addresses at first. As the number of necessary NotVia addresses for a node is much smaller than the total number of NotVia addresses, the gain of favoring the computation of the necessary ones can be significant. This is because only necessary entries matter for the restoration of the IPFRR service. If a new failure occurs before a router finishes computing all NotVia addresses, and all necessary NotVia entries have already been updated, packets encapsulated with NotVia addresses can still reach their destinations using the updated protection paths. Therefore, although prioritization does not reduce the total amount of NotVia computations, it improves the protection path's restoration time, thereby the network's reactivity to failures.

Closeness in our design is defined in terms of hop count, not cost metrics. This is because the metrics between core routers and access routers are often set to be very large in real ISP networks so that access routers do not provide transit service for other routers. The consequence is that prioritization based on the link costs will let a core router first compute NotVia entries for the protection of distant core links and routers. However, the core router has a low probability of actually being on the protection path of these links and routers. Instead, it has a high probability of being used to protect links or routers co-located at the same PoP, and those links and routers are closer to it than core links and routers in terms of hop count, not cost metrics.

A router needs an additional shortest path computation to obtain distance information in terms of hops (as the normal shortest path computation is cost based). But this overhead is negligible compared to the overhead of computing shortest path trees for all NotVia addresses. In our design, a router keeps a hop count based shortest path tree in memory so that it does not compute the hop count distances to all NotVia addresses at failure time but instead approximate the ordering using the topology before the change. After a topology change, the router can carry out an incremental shortest path first computation to compute the hop count based

Algorithm 1 How a node R computes the set of nodes that will use R on their protection paths to reach the NotVia address $NV_{\not{S}\to T}$ upon the failure of the node S. S is also included in this set if it uses R to send traffic towards $NV_{\not{S}\to T}$ upon the failure of $S \to T$.

```
Require: $G(E,V)$, link $S \to T$, router $R$
$G' \Leftarrow G - link(S \to T)$
for $N \in neighbor(S)$ do
    $G' \Leftarrow G' - link(N \to S)$
end for
$unreached\_nbr \Leftarrow S + neighbor(S)$
$upstream\_nbr \Leftarrow unreached\_nbr$
$flag(R)$
$maxdist \Leftarrow 0$
$finished \Leftarrow false$
while $not(finished)$ do
    $p \Leftarrow next\ path\ from\ the\ priority\ queue\ of\ rSPF(T, G')$
    $X \Leftarrow head(p)$
    if $unreached\_nbr = \emptyset$ and $cost(p) > maxdist$ then
        $finished \Leftarrow true$
    else
        if $X \in unreached\_nbr$ then
            if $cost(p) > maxdist$ then
                $maxdist \Leftarrow cost(p)$
            end if
            $unreached\_nbr \Leftarrow unreached\_nbr - X$
        end if
        if $flagged(nexthop(X, p))$ then
            $flag(X)$
        end if
    end if
end while
for $N \in upstream\_nbr$ do
    if $linkweight(N \to S) + linkweight(S \to T) > cost(N)$ then
        $upstream\_nbr = upstream\_nbr - N$
    end if
end for
return $\{U \in upstream\_nbr : flagged(U)\}$
```

distances, preparing the order of prioritization for the next topology change.

4. OBTAINING PROTECTION PATH INFORMATION

In this section, we describe the rNotVia algorithm which allows a router to efficiently compute the set of NotVia addresses it protects. This information is valuable for the purpose of network management, as described in Section 2. We first describe a strawman approach and analyze its computational cost. Then we present the more efficient rNotVia algorithm.

4.1 A Strawman Approach

A node R could obtain the information on whether it is on the protection path of a NotVia address $NV_{\not{S}\to T}$ by computing the shortest path tree rooted at a node that will encapsulate packets with the NotVia address $NV_{\not{S}\to T}$. There are two types of nodes that may use the NotVia address $NV_{\not{S}\to T}$: the head node S of the NotVia address $NV_{\not{S}\to T}$ to bypass the link $S \to T$ fails, and a node U one-hop upstream to the head node S to bypass S. By upstream, we mean a node U that uses S as the next hop to reach T. Therefore, a node would have to compute a shortest path tree rooted at each upstream node U on a topology excluding the head node S, and a shortest path tree rooted at the head node S excluding the link $S \to T$. Furthermore, a node R must know the set of nodes $\{U\}$ upstream to S to precisely determine whether it is on a protection path for the NotVia address $NV_{\not{S}\to T}$. This information needs to be precise for the purpose of estimating the amount of rerouted traffic when failures occur. However, the set $\{U\}$ is not generally available from the normal routing computation, as a node does not know other nodes' shortest paths. Obtaining this information may require an additional shortest path computation for each neighbor of the node S on the original topology without excluding any link or node. Therefore, a node R may perform up to $1 + size(\{U\}) + size(neighbor(S))$ rounds of shortest path tree computations to determine whether it is on the protection path of the NotVia address $NV_{\not{S}\to T}$.

4.2 The rNotVia Algorithm

We design an algorithm, rNotVia, that allows a node R to compute one single shortest path tree to determine whether it is on the protection path of a NotVia address $NV_{\not{S}\to T}$. This is a significant saving compared to the strawman approach. We describe the algorithm in three steps. In the first step, we describe how to combine multiple NotVia computations for the upstream nodes $\{U\}$ to bypass the node S into one shortest path computation. In the second step, we describe how to combine the NotVia computation to bypass the link $S \to T$ with that of bypassing the node S. Finally, we describe how a node can efficiently determine the upstream nodes $\{U\}$ without additional shortest path tree computations.

We can combine the multiple shortest path tree computations rooted at each upstream node U into a single shortest path tree computation by computing a reverse shortest path tree [2]. The reverse shortest path tree is constructed by using reverse shortest path first (rSPF) algorithm. For each NotVia address $NV_{\not{S}\to T}$, a node R computes a shortest path tree rooted at the node T with reversed link costs on the network topology excluding the node S. It can be shown that the shortest path from U to T on the network with normal link costs is the reverse path from T to U on the network with reversed link costs. Thus, this computation allows the node R to determine any upstream node U's protection path to reach the NotVia address $NV_{\cancel{S} \rightarrow T}$, and the cost of this reverse shortest path computation is the same as that of the normal shortest path.

The next step is to combine the shortest path tree computation for the link failure case with the node failure case. Link failure and node failure require separate computations because the shortest path trees are based on different network topologies. For the link failure case, the topology is the original network topology excluding the link $S \rightarrow T$, and for the node failure case, it's the topology excluding the node S.

To combine the shortest path tree computations for link and node failures, we modify the original network topology as follows. We remove all directed links that end at S, but keep all directed links that start from S except the link $S \rightarrow T$. On this modified topology, the shortest paths of all nodes other than S is the same as those in the topology with S removed, and the shortest paths of S is the same as that in the topology with the link $S \rightarrow T$ removed. Applying the reversed shortest path computation on this modified topology, we can obtain all shortest paths from all nodes upstream to S and S to reach the NotVia address $NV_{\cancel{S} \rightarrow T}$. Early termination and incremental shortest path first optimizations described in Section 2 can be applied to this computation as well to further speed up the computation.

Finally, we use a simple comparison to obtain the set of nodes $\{U\}$. For each neighbor N of S, if the sum of the costs of the two links $N \rightarrow S$ and $S \rightarrow T$ is less than or equal to the path cost from N to T in the topology excluding the node S, then N may use S as its next hop in the original topology. Thus N must be considered as an upstream node of S. The path cost from N to T in the topology excluding S can be obtained from the reverse shortest path computation described above. Thus, with at most one reverse shortest path computation, a node can precisely determine whether it is on the protection path of a NotVia address $NV_{\cancel{S} \rightarrow T}$. It is a significant saving in computation costs compared to the strawman approach, which requires up to $1 + size(\{U\}) + size(neighbor(S))$ rounds of shortest path tree computations.

We use a slightly revised version of Dijkstra algorithm (priority queue of the shortest paths instead of the nodes) to take care of equal cost multi-paths (ECMPs). In each round of the Dijkstra algorithm the current shortest path started from the source is pop-uped from the priority queue instead of the node with the smallest distance. The algorithm works correctly even if there are multiple protection paths from U to T with the same cost. The pseudo-code of rNotVia is presented by Algorithm 1, and its proof of correctness is given in Appendix B.

4.3 rNotVia Computational Cost

A node needs to compute a reverse shortest path tree for every NotVia address to determine the set of NotVia addresses it protects. We refer to this computation as the rNotVia computation. The total cost is on the order of $(E * SPT)$, where E is the number of directed links unprotected by LFA, and SPT is the computational cost of a shortest path tree. This cost is on the same order as the cost needed for a node to compute the NotVia address entries, i.e., the normal NotVia computation.

We have considered to replace the normal NotVia computation with the rNotVia computation. Both computations allow a node on the protection path of a NotVia address to compute the correct next hop, with rNotVia having the additional benefit to allow a node to determine to which NotVia addresses it is on the protection path. However, experimental results show that rNotVia computation is more expensive than the normal NotVia computation (Section 5.5). This is because the incremental shortest path tree optimization works more efficiently when a failure is far away from the root [17]. However, in the rNotVia computation case, the failure is always adjacent to the root of the shortest path tree. Therefore, in our design, we use the normal NotVia computation with aggregation and prioritization to compute NotVia address entries, and only use rNotVia computation as a background process to provide information to the Management Information Base.

5. EVALUATION

This section presents our evaluation on the practicality of the NotVia IP fast reroute scheme. We evaluate NotVia with and without our improvements, namely NotVia aggregation and prioritization. We also evaluate the efficiency of obtaining protection with the rNotVia algorithm.

5.1 Methodology

We implement a simulator that simulates the NotVia IPFRR scheme when a node or link fails in a network. We do not use ns-2 because it is extremely time consuming to collect all data we need. However, the results we obtain do not depend on simulators. They are determined by topology properties. The source code of our simulator is available at [1] for cross validation.

We run the simulator on five real ISP topologies as well as randomly topologies generated by a widely used topology generator BRITE [26]. The ISP topologies include real IGP link weights, and range from small regional ISPs to a worldwide tier-1 ISP. The number of nodes in those topologies are 20+, 50+, 100+, 200+, 400+ respectively, and the number of directed links are 50+, 200+, 400+, 700+, 1600+ respectively. Due to confidentiality, we cannot provide the exact numbers. For the BRITE topologies, we fix the number of nodes to 100, and vary other parameters. The random topologies are generated according to the Waxman model, the Barabasi-Albert (BA) and Barabasi-Albert-2 (BA2) model in BRITE.

For each topology, we collect the following statistics:

NotVia protection path restoration time. For each link or node failure, our simulator simulates the NotVia computations of each node in the topology. For each node, we record the total amount of time it takes for the node to finish updating its last necessary NotVia address (a NotVia address for which it is on the protection path). No other delays (FIB installation delay, failure detection delay, etc.) are introduced in the simulation because they are not affected by our proposals. We use the incremental shortest path first and early termination optimizations in computing the next hop for a NotVia address. We refer to this computation as the optimized SPT computation. The longest time among all nodes is used to estimate the network-wide NotVia protection path restoration time. We compare the results with and without prioritization and aggregation. In the case without prioritization and aggregation, we compute NotVia addresses in a random order. Partial aggregation and aggregation (Section 3) have the same amount of saving in the NotVia restoration time. Thus we do not differentiate them when presenting results on the protection path restoration time.

Number of NotVia forwarding table entries. We measure the number of forwarding table entries dedicated to NotVia addresses with aggregation, partial aggregation, and without aggregation. This metric is to show the memory overhead added by NotVia, and how aggregation and partial aggregation can reduce this overhead.

Number of NotVia announcements. For each link or node failure, we record the total number of NotVia announcements flooded after a topology change. NotVia announcements are used to signal which links or nodes are not protected by loop-free alternates after a topology change and vice versa. This metric does not directly relate to our improvements, but affects the protection restoration time of the NotVia IPFRR scheme. For stability reasons, routing protocol implementations limit the rate at which link state announcements can be fast flooded [8]. If there are too many NotVia announcements, they may not be rapidly flooded throughout the network. Delayed announcements, rather than NotVia computations, might become the bottleneck for protection restoration. Thus, we use this metric to evaluate whether NotVia computations are the bottleneck for protection restoration.

Protection information computation time. For each router and each NotVia address we simulate the time needed to compute whether the router is on the protection path of this address. This metric shows the efficiency of computing protection information for management purpose. In addition, to complement our claim that rNotVia is not suitable for computing NotVia entries because of its computational heaviness (Section 4.3), we also simulate the time needed to compute an entry for the address with optimized SPT computation and compare it to that with rNotVia.

For most experiments, we only show results for link failures. Node failures have similar results, and are omitted due to space limitations. Full set of results could be found in [24].

5.2 Protection Path Restoration Time

Figure 3 (a-c) shows the distributions of the number of optimized shortest path tree computations to compute all necessary NotVia entries after a link failure with and without prioritization and aggregation for three sample ISP topologies, and (d) shows the average on all five ISP topologies. Figure 4 shows the results for the BRITE topologies. For each type of BRITE topology, we choose a sample topology to show. All results are summarized in Table 3. As can be seen, prioritization and aggregation can significantly reduce the number of shortest path tree computations on both real ISP topologies and randomly generated topologies. The restoration time varies between different failures, this is because the effectiveness of prioritization is topology dependent. After a certain failure, if only few protection paths are affected, the restoration time is more likely to be short.

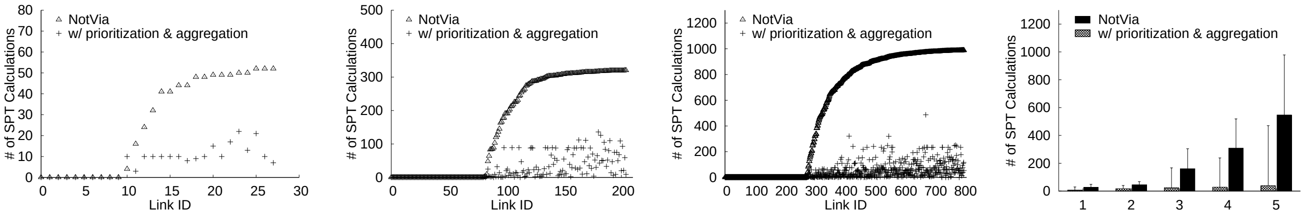
In Figure 3, many failures require zero computation to restore protection, because those links are not on any protection path. Thus, their failures do not alter the existing protection paths.

Figure 5 and 6 show the distributions of the time it takes to compute all necessary NotVia entries after a link failure with and without prioritization for real ISP topologies as well as randomly generated topologies. Those results provide insight on how time consuming NotVia can be without our optimization. The computation is done on a Pentium D 3.40GHz PC. Y-axes are the time measured in milliseconds and x-axes are link indices.

We note that the shapes of the distributions in terms of the number of shortest path tree computations (Figure 3 and 4) are different from that in Figure 5 and 6. This is because the runtimes of the optimized shortest path tree computations vary across nodes.

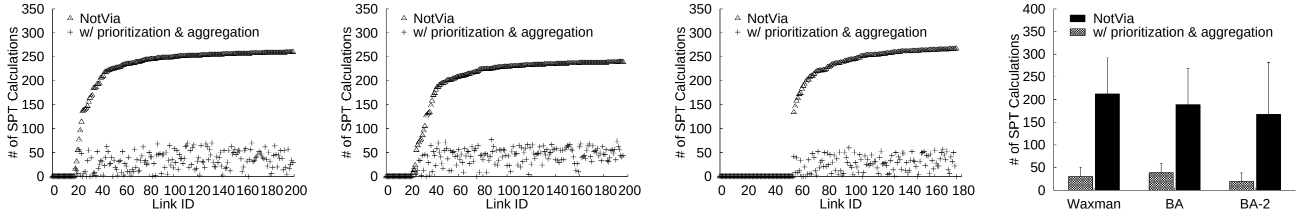
5.3 Forwarding Table Size

Figure 7 (a-c) shows the distributions of the number of NotVia forwarding table entries without aggregation, with partial and full aggregation on three sample ISP



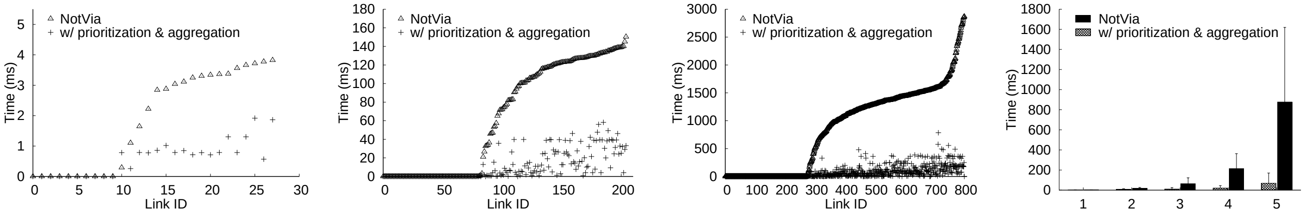
(a) 20+ Nodes (b) 100+ Nodes (c) 400+ Nodes (d) Avg. in five topolgies

Figure 3: **The distributions and average of the number of shortest path tree computations needed to finish computing all necessary NotVia addresses after a link failure. X-axes in (a-c) are failed link indices, and x-axis in (d) is the topology index. Y-axes are the number of the optimized shortest tree computations, and are in different ranges for clarity. The vertical lines in (d) show the standard deviations.**



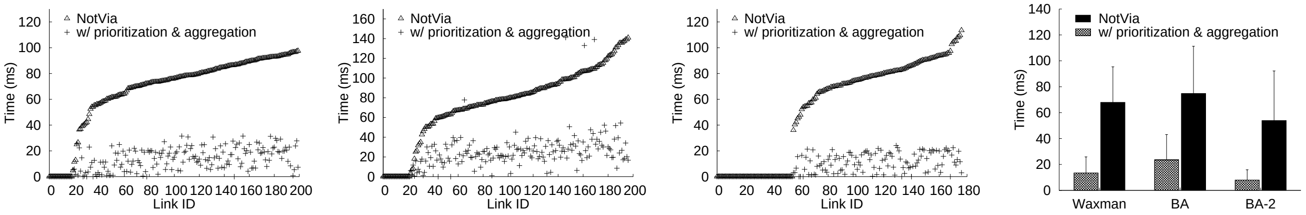
(a) Waxman (b) Barabasi-Albert (c) Barabasi-Albert-2 (d) Avg. in three topologies

Figure 4: **The same results as in Figure 3 for the random topologies.**



(a) 20+ Nodes (b) 100+ Nodes (c) 400+ Nodes (d) Avg. in five topolgies

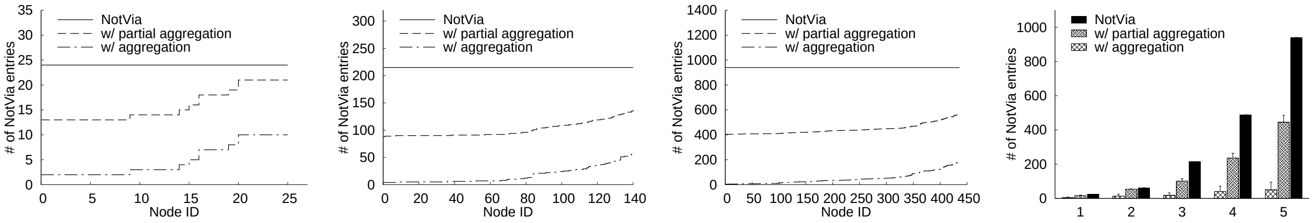
Figure 5: **The distributions and average of the time needed to finish computing all necessary NotVia entries after a link failure on our test PC. This time affects how quickly NotVia protection paths can be restored. X-axes in (a-c) are failed link indices, and x-axis in (d) is topology index. Y-axes are in unit of milliseconds, and are in different ranges for clarity. The vertical lines in (d) show the standard deviations. Prioritization and aggregation reduce the amount of time to restore the protection paths.**



(a) Waxman (b) Barabasi-Albert (c) Barabasi-Albert-2 (d) Avg. in three topologies

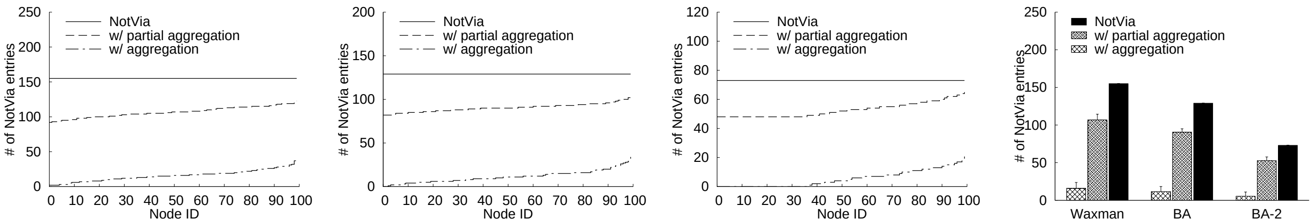
Figure 6: **The same results as in Figure 5 for the random topologies.**

topologies, ranging from the smallest one to the largest one. Figure 8 shows the same results for the randomly generated topologies. Y-axes are the number of NotVia entries, and x-axes are node indices. Both the results with aggregation and with partial aggregation reduce the number of NotVia entries, especially in the large tier-1 ISP network. The value with aggregation is only a small fraction of the one without aggregation. Figure 7(d) and 8(d) show the average number of NotVia entries for the five ISP topologies and the randomly gen-



(a) 20+ Nodes (b) 100+ Nodes (c) 400+ Nodes (d) Avg. in five topologies

Figure 7: **The distributions and average of the number of NotVia forwarding table entries. X-axes in (a-c) are router indices, and x-axis in (d) is the topology index. Y-axes are the number of NotVia entries. They are shown in different ranges for clarity. Both aggregation and partial aggregation reduce NotVia's memory overhead, with aggregation being more effective.**



(a) Waxman (b) Babarasi-Albert (c) Babarasi-Albert-2 (d) Avg. in three topologies

Figure 8: **The same results as in Figure 7 for the random topologies.**

Topology	Avg. # of NotVia LSAs	
	Link Failure	Node Failure
ISP-1	0.07	0.08
ISP-2	0.80	0.81
ISP-3	0.44	0.54
ISP-4	0.87	1.44
ISP-5	1.27	1.00
Waxman	0.01	0.02
BA	0.01	0.02
BA-2	0.003	0.005

Table 2: **Average numbers of Notvia link state announcements announced by all routers after a failure on all five ISP topologies and three randomly generated topologies.**

erated topologies respectively. As can be seen, aggregation and partial aggregation can reduce the number of NotVia entries in all cases.

5.4 Number of NotVia Announcements

Table 2 shows the average number of NotVia link state announcements originated by all routers in the network after a node or a link failure. The average is taken over all link or node failures. As can be seen, very few NotVia announcements are originated, indicating that those announcements can be rapidly flooded throughout the network. A router can receive all new NotVia announcements before it starts NotVia computations. Thus flooding NotVia link state announcements is unlikely to be the bottleneck to restore NotVia protections, and aggregation and prioritization will be useful in improving the protection restoration time.

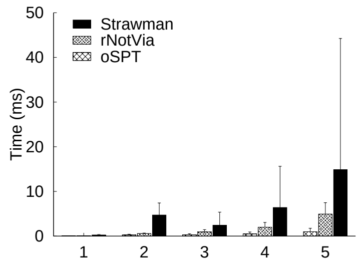


Figure 9: **Average time needed for a router to compute whether it is on the protection path of a NotVia address. X-axis is the topology index, and y-axis is the time in the unit of milliseconds. rNotVia requires much less time than the strawman approach.**

5.5 Efficiency of rNotVia

Figure 9 shows the average time for a router to compute the protection information for a NotVia address, i.e., to examine whether it is on the protection path of the address. The average is taken over all routers and all NotVia addresses in the network. We can see that rNotVia requires much less time than the strawman approach. "oSPT" in the figure shows the average time for a router to compute a NotVia entry with optimized SPT algorithm. As can be seen, rNotVia is generally heavier than optimized SPT because of the reason we explained in Section 4.3. Thus we only propose to use rNotVia for management purpose. Results with random topologies are similar and omitted due to space limitation.

	Topology	# of Nodes	# of Links	Restoration Time(ms)		Number of SPTs		Number of NotVia Entries		
				Optimized	Current	Optimized	Current	w/ Aggregation	w/ Partial	Current
Real	ISP-1	20+	50+	0.6	1.9	7.3	26.6	5.03	16.0	24.0
	ISP-2	50+	200+	7.9	17.2	16.7	44.4	13.4	53.6	61.0
	ISP-3	100+	400+	10.8	64.2	23.3	160.6	17.2	101.1	215.0
	ISP-4	200+	700+	20.4	214.5	28.3	308.8	40.1	236.1	488.0
	ISP-5	400+	1600+	68.6	876.7	39.0	547.0	49.9	445.0	940.0
Random	Waxman-1	100	400	13.4	68.0	30.1	212.7	15.8	106.9	155.0
	BA-1	100	394	23.5	74.8	38.2	188.8	11.5	90.4	129.0
	BA2-1	100	354	8.0	53.9	19.1	167.5	5.4	52.7	73.0
	Waxman-2	100	600	8.2	46.8	18.2	129.9	15.5	80.0	102.0
	BA-2	100	588	18.7	67.8	28.8	140.4	12.7	61.4	72.0
	BA2-2	100	762	8.4	40.0	13.4	78.1	6.5	34.4	37.0

Table 3: Summary of Simulation Results on All Topologies

5.6 Summary

Table 3 summarizes the results on all topologies, including the average protection restoration time and the average forwarding table entries dedicated to NotVia addresses. As can be seen, in all topologies, the proposed aggregation and prioritization techniques reduce the computational overhead as well as memory overhead of NotVia.

6. RELATED WORK

Francois et al. [13] and Gjoka et al. [16] evaluated the failure coverage of different IPFRR solutions, while this work improves and evaluates the specific technique NotVia.

Other work [3,5,22,27] provides different IPFRR solutions. Among them, Uturn [3] and tunnels [5] do not provide full coverage for single failures. Failure Insensitive Routing [27] provides full coverage for single failures on symmetric networks, and requires interface-specific forwarding tables. Multiple topology configurations (MRC) [22] are similar in spirit to NotVia addresses. Each topology configuration is computed by removing protected links or nodes from the original topology. Different from NotVia, a topology configuration may remove multiple links and nodes. MRC is able to use the same topology configuration to protect multiple links, thereby reducing the number of forwarding table entries to store protection paths. However, its worst case computational overhead is even higher than NotVia.

FCP [23] explores a design point that assumes packets can carry failure information in their headers and routers can perform on demand shortest path computations based on the failure information. With these assumptions, routers can fast reroute packets around multiple failures without requiring consistent routing state. The NotVia scheme we explore does not require on demand shortest path computations, and does not alter the IP header format.

7. CONCLUSION

This paper presents techniques that reduce the overhead of the NotVia IP Fast Reroute scheme. NotVia is currently being debated at IETF as a candidate for standardization. However, the computational and memory overhead of NotVia have cast some doubts on its practicality. Our techniques, NotVia aggregation and prioritized NotVia computation, are based on the simple intuitions that protection paths computed for NotVia addresses may overlap with normal forwarding paths, and are also likely to be local to the failures. We evaluate these techniques on real ISP topologies ranging from small ISPs to a large tier-1 ISP and randomly generated topologies. The results show that our techniques can effectively reduce the NotVia protection path restoration time and the forwarding table entries dedicated to NotVia addresses, especially on large ISP networks. We also present an algorithm rNotVia that allows a router to efficiently determine whether it is on the protection path of a NotVia address. This information can be used by network operators to estimate the amount of rerouted traffic and to configure their networks from being overloaded by the rerouted traffic. We believe that the techniques presented in this paper can significantly improve the efficiency and manageability of the NotVia IPFRR scheme, and this work is a step further towards an efficient and easy-to-manage IPFRR solution.

Acknowledgements

We would like to thank the anonymous reviewers for their comments, and Geoffroy Jennes, Frédéric Grignet, Benoit Fondeviolle and Nicolas Simar for their help in collecting ISP topologies.

8. REFERENCES

[1] Project Website. http://nds.ics.uci.edu/notvia.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *NETWORK FLOWS: THEORY, ALGORITHMS, AND APPLICATIONS*. Prentice Hall, 1993.

[3] A. Atlas. U-turn alternates for IP/LDP Fast-Reroute. Internet draft, draft-atlas-ip-local-protect-uturn-03.txt, Feb 2006.

[4] A. Atlas and A. Zinin. Basic specification for IP fast-reroute : Loop-free Alternates. Internet draft, draft-ietf-rtgwg-ipfrr-spec-base-06.txt, Feb 2006.

[5] S. Bryant, C. Filsfils, S. Previdi, and M. Shand. IP Fast Reroute using tunnels. Internet draft, draft-bryant-ipfrr-tunnels-02.txt, Apr 2005.

[6] S. Bryant and M. Shand. A framework for loop-free convergence. Internet draft, draft-bryant-shand-lf-conv-frmwk-03.txt, Oct 2006.

[7] S. Bryant, M. Shand, and S. Previdi. IP fast reroute using notvia addresses. Internet draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-00.txt, Dec 2006.

[8] Cisco. IS-IS Fast-Flooding of LSPs Using the fast-flood Command. Technical document, `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s27/fstfld.pdf`, 2003.

[9] N. Feamster and H. Balakrishnan. Packet Loss Recovery for Streaming Video. In *International Packet Video Workshop*, 2002.

[10] S. Fischer, N. Kammenhuber, and A. Feldmann. REPLEX — dynamic traffic engineering based on wardrop routing policies. In *CoNext*, 2006.

[11] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM (2)*, pages 519–528, 2000.

[12] P. Francois and O. Bonaventure. Avoiding transient loops during IGP convergence in IP networks. In *IEEE INFOCOM*, Miami, Florida, USA, Mar 2005.

[13] P. Francois and O. Bonaventure. An evaluation of ip-based fast reroute techniques. In *Co-Next*, 2005.

[14] P. Francois, O. Bonaventure, M. Shand, S. Previdi, and S. Bryant. Loop-free convergence using ordered FIB updates. Internet draft, draft-francois-ordered-fib-01.txt, Mar 2006.

[15] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *SIGCOMM Comput. Commun. Rev.*, 35(3):35–44, 2005.

[16] M. Gjoka, V. Ram, and X. Yang. Evaluation of IP fast reroute proposals. In *IEEE Comsware*, 2007.

[17] W. J. Goralski and H. Gredler. *The Complete IS-IS Routing Protocol*. Springer, 2005.

[18] J. He, M. Bresler, M. Chiang, and J. Rexford. Towards multi-layer traffic engineering: Optimization of congestion control and routing. *IEEE Journal on Selected Areas in Communications*, 2007.

[19] G. Iannaccone, C. Chuah, S. Bhattacharyya, and C. Diot. Feasibility of IP restoration in a tier-1 backbone. *IEEE Network Magazine*, Jan-Feb 2004.

[20] W. Jiang and H. Schulzrinne. Comparison and Optimization of Packet Loss Repair Methods on VoIP Perceived Quality under Bursty Loss. In *NOSSDAV*, 2002.

[21] D. Katz, K. Kompella, and D. Yeung. Traffic engineering (te) extensions to ospf version 2. Technical report, Internet Engineering Task Force, United States, 2003.

[22] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast ip network recovery using multiple routing configurations. In *Infocom*, 2006.

[23] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, , S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *ACM SIGCOMM*, 2007.

[24] A. Li, P. Francois, and X. Yang. On Improving the Efficiency and Manageability of NotVia. Technical report, `http://www.ics.uci.edu/~angl/papers/notvia_report.pdf`.

[25] K. McCloghrie and M. T. Rose. Structure and identification of management information for TCP/IP-based internets. Request for Comments 1065, Internet Engineering Task Force, Aug. 1988.

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

[27] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Trans. Netw.*, 2007.

[28] rtgwg. IETF routing area working group meeting minutes. http://www3.ietf.org/proceedings/06nov/minutes/rtgwg.txt, Nov 2006.

[29] M. Shand and S. Bryant. IP Fast Reroute Framework. Internet draft, draft-ietf-rtgwg-ipfrr-framework-06.txt, Oct 2006.

[30] R. Sridharan and C. Diot. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks, 2002.

[31] J.-P. Vasseur, M. Pickavet, and P. Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, and MPLS*. Morgan Kaufmann, 2004.

[32] A. Zinin. Analysis and minimization of microloops in link-state routing protocols. Internet draft, draft-zinin-microloop-analysis-00.txt, Oct 2004.

APPENDIX

A. PROOF FOR CORRECTNESS OF PROTECTION PATHS

Theorem A.1 *Let R be a node in the network. If R is on any protection path of link $S \to T$, and R is not S, the next hop calculated based on the topology removing the link $S \to T$ is the same as the one calculated removing the node S.*

Proof. Let $cost_{S \nrightarrow T}(X, Y)$ be the shortest path length from node X to Y removing the link $S \to T$. As R is on the shortest path from S to T removing the link $S \to T$, and R is not S, we have $cost_{S \nrightarrow T}(R, T) < cost_{S \nrightarrow T}(S, T)$. Suppose S is on the shortest path from R to T removing the link $S \to T$, we have $cost_{S \nrightarrow T}(R, T) > cost_{S \nrightarrow T}(S, T)$, which contradicts with our previous result. Hence S is not on any shortest path from R to T removing the link $S \to T$. Therefore, removing the node S from the topology will not affect the shortest paths from R to T. In other words, R's next hop towards T without the link $S \to T$ is the same as the one without the entire node S. □

B. PROOF FOR CORRECTNESS OF RNOTVIA

Theorem B.1 *Let R be a node, and $S \to T$ be any directed link in the network. After running rNotVia based on $S \to T$ and R, the returned result contains all nodes which will encapsulate traffic with $NV_{\not S \to T}$ and send the packets via R.*

Proof. We prove by analyzing the pseudo-code presented in Algorithm 1.

1. At line 4, G' is equal to G where the links from the neighbors of S to S and the link $S \to T$ have been removed.
2. From 1, a shortest path from a node $N \neq S$ to T in G' does not contain S as there is no link whose tail-end is S in G'.
3. From 2, the shortest path from a node $N \neq S$ to T in G' is the shortest path from N to $NV_{\not S \to T}$, according to the definition of $NV_{\not S \to T}$.
4. From 2, the shortest path from S to T in G' is the shortest path from S to $NV_{\not S \to T}$, according to the definition of $NV_{\not S \to T}$ when as used by S for link protection.
5. At line 8, R is the only flagged node
6. At line 8, *unreached_nbr* and *upstream_nbr* contain the nodes that may send traffic towards $NV_{\not S \to T}$
7. The loop starting at line 10 terminates when *unreached_nbr* is empty and the next shortest path provided by the priority queue of the rSPT computation is longer than any path from a node in *upstream_nbr* to T
8. From 7, the loop terminates when all the path from the neighbors of S and S to T in G' have been extracted from the priority queue.
9. During the loop starting at line 10, the head of a path to T is flagged only if its next hop is flagged.
10. From 9 and 5, a node is flagged only if one of its shortest paths to T in G' contains R.
11. From 8 and 10, each neighbor of S and S is flagged if its shortest paths to T in G' contain R
12. The loop starting at line 27, removes the nodes in *upstream_nbr* that are not using S to reach T in G. They will not send packets with destination $NV_{\not S \to T}$ when S fails . S is removed from *upstream_nbr* if it will not send packets with destination $NV_{\not S \to T}$ when $S \to T$ fails, i.e. $S \to T$ is not used
13. From 12 and 11, the returned set of nodes contains the nodes that will send packets with destination $NV_{\not S \to T}$ via R upon the failure of S or $S \to T$.

□