IP Fast Reroute: NotVia with Early Decapsulation

Qing Li*, Mingwei Xu*, Qi Li*, Dan Wang† and Yong Cui*
* Dept. of Computer Science, Tsinghua Univ., Tsinghua National Laboratory for Information Science and Technology
† Dept. of Computing, The Hong Kong Polytechnic Univ.
Email: {liqing, xmw, liqi, cuiyong}@csnet1.cs.tsinghua.edu.cn, csdwang@comp.polyu.edu.hk

***Abstract*—Network survivability is an important topic for the Internet. To improve the performance of the Internet during failure, IP Fast Reroute (IPFRR) mechanisms are proposed to establish backup routes for failure-affected packets. NotVia, a most prominent one, provides 100% protection coverage for single-node failures. However, it brings in nontrivial computing and memory pressure to routers with special NotVia addresses, in which only some are necessary for a specific router. Besides, the protection path of NotVia is 20% longer than the optimal path on average.**

In this paper, we propose early decapsulated NotVia (ED-NotVia) handling the aforementioned problems and thus making NotVia more practical. We first analyze the properties of necessary NotVia addresses to any specific node. Then we develop a heuristic Nec-NotVia Algorithm for a node to find the necessary NotVia addresses and compute routes for them, where unnecessary addresses are eliminated. Based on this elimination, early decapsulation is imported to optimize the protection path with marginal overhead. We evaluate our algorithm and demonstrate the effectiveness of ED-NotVia using topologies from Rocketfuel and Brite. The results show that 1) only 5% to 20% of SPT(Shortest Path Tree)-related NotVia addresses (1.23% to 6.41% of all the NotVia addresses) in an AS are necessary for a node; 2) by computing the routes for 15% to 40% SPT-related NotVia addresses, ED-NotVia provides 98% protection coverage; and 3) the protection path stretch ratio of ED-NotVia is only 1.03 on average as compared to 1.20 for NotVia.

I. Introduction

In order to provide high-quality transport service for all kinds of web applications, IP routing is thus designed for robust operation in case of changing network states. However, these IP routing protocols take on the order of a few hundred milliseconds or even tens of minutes to re-converge [1–3]. Recent large-scale deployments of delay and loss-sensitive applications, such as (VoIP), streaming media, gaming, and telecommuting/video, have led to stringent demands on the Internet routing [4]. ISPs hence have strong incentives to improve the Internet survivability.

IPFRR can provide resilience in the event of a failure by quickly (less than 50ms) rerouting traffic to pre-computed alternates. By computing backup routes beforehand and handling failures locally, IPFRR can reduce the disruption time as short as the failure detection time and meanwhile suppress transient failures to improve routing stability [5–7].

However, current IPFRR mechanisms cannot meet ISPs' requirements. Loop-free alternate (LFA) [8] is lightweight, but it provides only 30-40% protection coverage. Tunnel [9] provides a high protection coverage but requires too much computing to find tunnel ends. Compared with LFA and Tunnel, NotVia [10] can provide 100% protection coverage for single-node failures. However, it meanwhile brings nontrivial computing and memory overhead to maintain a set of routes for special NotVia addresses.

A practical IPFRR mechanism has two important features: first, it should be lightweight: bring in as little overhead as possible and react rapidly to failures; second, it should have an excellent performance including a higher protection coverage and a shorter protection path. In this paper, we propose ED-NotVia, an intra-domain IPFRR mechanism, which meets these two requirements well.

We analyze the properties of necessary NotVia addresses to any specific node R. According to our analysis, only those special and related (see below in section III) NotVia addresses are necessary to R. According to the properties, we propose a heuristic Nec-NotVia Algorithm for R to find those necessary NotVia addresses and compute routes for them, so than unnecessary NotVia addresses are completely eliminated. Based on this elimination of unnecessary NotVia address, early decapsulation is introduced to optimize the protection path with marginal overhead.

An Example. Before exposition, we illustrate our ED-NotVia using an example in Fig. 1. In total, eighteen (as the same as the number of unidirectional links [6]) NotVia addresses exist in the small AS. To R, only T_F (to T not via F), D_G and H_T are necessary. If F fails, in NotVia the protection path from S to D is $S{\rightarrow}R{\rightarrow}G{\rightarrow}H{\rightarrow}T{\rightarrow}H{\rightarrow}D$. With early decapsulation introduced, ED-NotVia has a much shorter protection path $S{\rightarrow}R{\rightarrow}G{\rightarrow}D$. Because $route(G, D)$ is not via F , G can decapsulate the received T_F packets and forward them by its ordinary Forwarding Information Base (FIB). G makes the decision of early decapsulation according to the fact that T_F does not exist in its NotVia FIB.

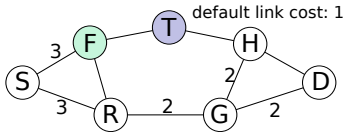


Fig. 1. An example of ED-NotVia.

The above example is not special; from analysis and experiments, we find that the ratio of necessary NotVia adddresses is quite small, especially for large topologies. We demonstrate the effectiveness of ED-NotVia by simulation based on topologies from Rocketfuel [11] and Brite [12]. The results show:

- Only 5%-20% of SPT-related NotVia addresses (1.23%-6.41% of all the NotVia addresses) are necessary. With all the unnecessary ones eliminated, the average amount

of NotVia addresses for each node has an upper limit of about 40 no matter how large the AS is.
- The proposed algorithm Nec-NotVia makes a tradeoff between protection coverage and computing overhead by configuring parameter, *ratio-p*. By computing the routes for 15% to 40% SPT-related NotVia addresses, ED-NotVia provides 98% protection coverage.
- With early decapsulation adopted, the protection path stretch ratio of ED-NotVia is only about 1.03 (compared with 1.20 for NotVia) on average, which means it alleviates NotVia's protection path stretch by 85%.

The remaining part of the paper is organized as follows. In the next section, we further describe NotVia and some improvements over it. And then, in Section III, we introduce the details of ED-NotVia. Section IV involves the evaluation results, which show the performance of ED-NotVia. Finally, we conclude our paper in Section V.

II. Background and Related Work

When a failure occurs, the failure adjacent routers cannot tell it is a node or link failure. In NotVia, to cope with the worst case, the adjacent router (or the protection source node) would regard it as a node failure and attempt to find a protection path for it. Fig. 1 shows an example of NotVia. F is the failed node, S is upstream node of F and T is the next next hop (NNHOP) of $route(S, D)$. If F fails, then the protection source node (PSN) S would encapsulate the packet to D using a special NotVia address T_F (to T not via F). When T receives this packet packet, T decapsulates it and routes it to D. The protection path is $S{\rightarrow}R{\rightarrow}G{\rightarrow}H{\rightarrow}T{\rightarrow}H{\rightarrow}D$. Compared with other mechanisms such as LFA [8], U-Turn [13] and Tunnel [9], NotVia is excellent that it provides 100% protection coverage for single-node failures. However, some serious limitations block it from practical deployment.

As shown in Fig. 1, nodes along the protection path from S to T have to keep an entry for the address T_F in their NotVia FIBs. As there is no efficient way for a node (R for example) to tell whether itself is on the protection path of T_F, R has to maintain an entry for T_F. And the amount of T_F-like NotVia addresses in an AS is the same as the number of the directed links [6], introducing nontrivial computing and memory overhead. Besides, the protection path contains a transient loop (H, T, H), increasing the protection path stretch of NotVia. In fact, when a T_F packet arrives at G, the failure has already been bypassed and G can decapsulate this T_F packet.

There are some works that focus on improving NotVia's performance. In [6], Li, *et al.*, introduce the NotVia aggregation and prioritization techniques to reduce the computing costs and the forwarding table entries dedicated to maintaining the NotVia state. However, the aggregation approach can reduce only part of the unnecessary NotVia FIB entries and the algorithm they propose cannot guarantee protection coverage. In [14], Enyedi, *et al.*, propose lightweight NotVia using the concept of redundant trees [15]. Lightweight NotVia significantly decreases the number of Not-via addresses. However, in this mechanism, the PSN requires to forward two copies of one packet to NNHOP by two different routes, and other nodes along the two different protection paths has to forward the received packets by different routes too. It is possible that one of the two copies will not arrive at NNHOP and it is also possible that both of the copies will arrive at NNHOP. This approach would aggravate network congestion during failure by wasting bandwidth. And if NNHOP receives both of the two copies, it should drop one after the decapsulation (a record is required). To be brief, light NotVia does an excellent job on decreasing NotVia addresses, but it also introduces new problems.

III. ED-NotVia

In this section, we first discuss the elimination of unnecessary NotVia entries and the early decapsulation of NotVia packets separately, and then provide a practical and novel approach to combine them together. Finally, we show our algorithm for any node in an AS to find all necessary NotVia addresses and compute backup routes for them.

A. NotVia FIB Shrink

Without any improvement, the number of entries in NotVia FIB is equal to the number of unidirectional links in the AS [6], which is intolerable. In this part, we propose the *Condition RS* (related and special), according to which NotVia FIB of R can be shrunk completely.

Definition 1. T_F is special to R: if the next hop of $route(R, T_F)$ is the same with the next hop of $route(R, T)$, T_F is ordinary to R; else, T_F is special to R.

If T_F is ordinary to R, it is obvious that R does not require T_F entry. When receiving a T_F packet, R can forward it according to T entry in its general FIB. (Here we assume that T is inferrable from T_F. Actually our mechanism does not require this inference, which we will explain later.) However, the fact that T_F is special to R does not mean R requires T_F. For example, in Figure 2, T_F is special to R, but T_F is unnecessary for R, because R never receives T_F packets if F actually fails.

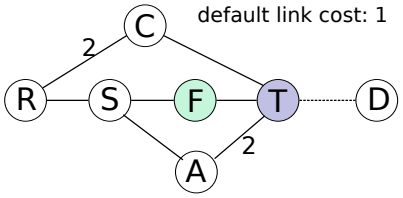


Fig. 2. T_F is special to R, but R does not require T_F entry.

Definition 2. T_F is related to R: if for each PSN S, $route(S, T_F)$ is not via R, T_F is unrelated to R; else, T_F is related to R.

According to the above definitions, we conclude that T_F is necessary for R if and only if T_F is related and special (*Condition RS*) to R. If T_F is related to R, R will receive T_F packets when F actually fails; meanwhile, if T_F is special

to R, R cannot forward T_F packets to T without T_F entry. Therefore, if a NotVia address satisfies *Condition RS*, R computes a route and save a NotVia entry for it; otherwise R does nothing for it. When receiving a T_F packet, R forwards it by the NotVia entry if the entry exists in its NotVia FIB; if not, R forwards the packet by T entry in its ordinary FIB.

B. Early Decapsulation

Before delving into the details of early decapsulation of NotVia packets, we review the protection path stretch and decapsulation bottleneck problems of NotVia. Note that if $route(R, T_F)$ and $route(T, D)$ have some common nodes except T, there will be a transient loop in the protection path [10]. For example, in Fig. 3, the protection path is $S{\rightarrow}X{\rightarrow}Y{\rightarrow}Z{\rightarrow}A{\rightarrow}T{\rightarrow}A...D$, which is not the best choice. Besides, T requires to decapsulate all the T_F packets, which is a bottleneck of forwarding speed. For example, in Fig. 3, T_F packets encapsulated by S, B and P are all decapsulated by T, which may incur nontrivial overhead on T.

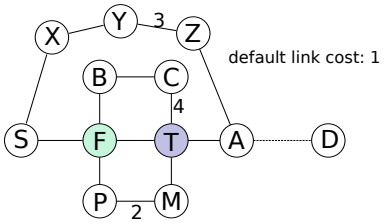


Fig. 3. Protection path stretch and decapsulation bottleneck of NotVia

However, both the above problems can be solved by *early decapsulation* of NotVia packets. As shown in Fig. 3, if Y or M receives a T_F packet, they can decapsulate the packet and forward it according to their ordinary FIBs. Because both $route(Y, D)$ and $route(M, D)$ are not via F, which means that the failure has been bypassed when the protection packet arrives at Y or M. If early decapsulation is adopted, the T_F packets from S and P would be decapsulated before reaching T, and only the T_F packets from B still require T to decapsulate. Protection path stretch gets alleviated and decapsulation bottleneck problem does not exist any more.

C. ED-NotVia Framework

In this part, we provide a practical and novel approach to combine NotVia FIB shrink and early decapsulation together.

Once R makes the decision of early decapsulating a T_F packet by checking whether F is on $route(R, D)$, R requires to get the destination D in the payload of the T_F packet and compute $route(R, D)$, which affects forwarding speed and meanwhile is unpractical because of the computing overhead. Here, we propose another approach for R to determine whether to decapsulate a received T_F packet. Still assuming that F and T are inferrable from the NotVia address T_F, we provide the below condition for R to early decapsulate a T_F packet:

$$cost(R, T) < cost(R, F) + cost(F, T)$$

The condition means that $route(R, T)$ is not via F, from which we can infer that $route(R, D)$ is not via F either:

$$\begin{aligned} cost(R, D) &\leq cost(R, T) + cost(T, D) \\ &< cost(R, F) + cost(F, T) + cost(T, D) \\ &= cost(R, F) + cost(F, D) \end{aligned}$$

Note that the condition that $route(R, T)$ is not via F is a similar to the condition that T_F is ordinary to R. The former means $route(R, T)$ and $route(R, T_F)$ are same, and the latter means they have the same next hop. According to this, we combine NotVia FIB shrink and early decapsulation together.

Assuming that R has shrunk its NotVia FIB according to *Condition RS*, when receiving a T_F packet, R processes it as follows (*this approach avoids using T or F to make the decision of early decapsulation*):

1. If the packet is a NotVia protection packet, go to step 3; else go to step 2.
2. Forward the packet according to the ordinary FIB.
3. Look up the NotVia FIB, if the prefix does not exist, decapsulate it and switch to step 2; else, forward the packet by the entry found in the NotVia FIB.

Mostly, this approach works with no problem. Although this may introduce the re-encapsulation problem, it would not cause loops. For example, in Fig. 4, T_F is ordinary for R, and R does not save T_F entry in its NotVia FIB. When R receives a T_F packet (the original destination is D) from S, R decapsulates it and forwards the D packet to S according to its ordinary FIB. Once S receives this packet, it will encapsulate it with T_F address again if F actually fails.

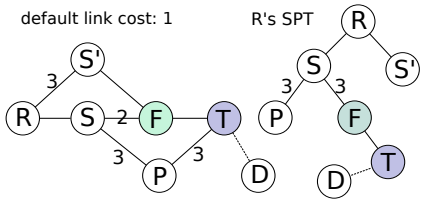


Fig. 4. An example of re-encapsulation

There are two ways to handle the re-encapsulation problem:

- Make Definition 1 a little more stringent: T_F is ordinary to R if and only if $route(R, T_F)$ is the same with $route(R, T)$ (not just the next hop). With a very few extra NotVia entries added in the NotVia FIB, the re-encapsulation problem does not exist any more.
- Ignore this problem: the probability of re-encapsulation is very low in real situation. Because if it is the link between S and F (not the node F) that fails, the re-encapsulation problem does not exist either; and more than 70% of failures are single-link failures [16].

Both approaches are acceptable and we believe how to choose is an implementation-specific issue. In this paper, we choose the second one because of the following three reasons. First, as we mentioned above, the re-encapsulation probability is very low in real application scenarios. Most of decapsulation would not cause re-encapsulation. Second, the second method can provide more complete NotVia Forwarding Information Base (FIB) shrink. Third, meanwhile the most important, early decapsulation optimizes the protection path

if it is a link failure (more than 70% of failures are single-link failures [16]). For example in Fig. 4, if $link(S', F)$ fails, without early decapsulation the protection path for packets to D from S' would be $S'{\rightarrow}R{\rightarrow}S{\rightarrow}P{\rightarrow}T...D$ (cost: 10); by introducing early decapsulation, the protection path is $S'{\rightarrow}R{\rightarrow}S{\rightarrow}F{\rightarrow}T...D$ (cost: 7).

We have shown the major structure of ED-NotVia and the remaining problem is to design an algorithm for R to find those necessary NotVia addresses and compute routes for them.

D. Nec-NotVia Algorithm

1) Key Protection Source Node: If T is not the child of F in the SPT of R, then T_F is ordinary to R. Therefore, we ignore this kind of unnecessary NotVia addresses and only focus on those *SPT-related* NotVia addresses.

To determine whether T_F is related to R, R requires to verify whether itself is on any of the protection paths from all PSNs (all F's neighbors excluding T) to T_F, with each protection path requiring one time SPT calculation. In this part, we give the definition of key-PSN and prove that: if the route from key-PSN to T_F is not via R, R does not require the T_F entry. With early decapsulation, this simplification can still guarantee that the T_F packet will be routed to the destination without loops and mostly the protection path would be optimized closer to the shortest path.

Definition 3. To R, the key-PSN (key protection source node) for T_F is the upstream node of F in the SPT of R.

Assuming that S is the key-PSN and S' is any other general PSN for T_F (for example, in Fig. 4), if R is on $route(S, T_F)$, R saves T_F entry and no problem comes out; otherwise, R does not save T_F entry. And when receiving a T_F packet, R would decapsulate it and restore its original destination. Actually, three possibilities exist after the decapsulation:

- $route(R, D)$ is not via F. So early decapsulation is the best choice, which is proved in the above subsection.
- F is on $route(R, D)$, but it is the link between S' and F (not the node F) that fails. If so, the early decapsulation is also the best choice for the packet. For example, in Fig. 4, if D is the original destination and $link(S', F)$ fails, the T_F packets from S' are decapsulated by R and the protection path is optimized a lot.
- $route(R, D)$ is via F and F actually fails. If so, after the early decapsulation, the packet would be routed to S, which is the key-PSN of R for T_F. And then S would re-encapsulate it as a T_F packet. Anyway, the packet would reach the destination. Fig. 4 is a specific example.

For the worst case, this simplification causes nothing more than re-encapsulation, which we have proved is not a big issue. In fact, 70% of the failures are single-link failures, only 16.5% are single-node failures, and others are multiple failures [16]. Therefore, in most cases, this simplification not only makes it easier for R to decide whether T_F is related to itself but also optimizes protection paths.

2) Algorithm Design: According to the discussion above, R can decide whether T_F is necessary to itself by verifying whether the route from key-PSN to T_F is via itself.

Intuitively, the further F is from R, the less possible T_F is necessary for R. Assuming that $route(R, F)$ is $R{\rightarrow}R_k{\rightarrow}R_{k-1}...R_2{\rightarrow}R_1{\rightarrow}F$ and d_i is the degree of R_i, if R receives the T_F packet from R_1 (key-PSN), it means $route(R_1, T_F)$ is $R_1{\rightarrow}R_2...R_{k-1}{\rightarrow}R_k{\rightarrow}R...T_F$. We suppose the probability that R_i forwards a T_F packet to R_{i+1} is $1/d_i$., then the probability that R receives the T_F packet from R_1 is $1/\prod d_i$. So a threshold *ratio-p* (p for short) can be set to guarantee the protection coverage. If $1/\prod d_i < 1 - p$, R would not compute a route for T_F, because the possibility that *route(key-PSN, T_F)* traverses R is low enough. For example, assuming that p is set to 0.95, when $\prod d_i > 20$, the possibility that T_F is necessary to R is less than 0.05, and then R would not save the T_F entry.

Algorithm 1 Nec-NotVia

Input: a node R, the SPT of R $\mathcal{T}$ and parameter p
Output: the routes of necessary NotVia addresses

if $0 < p < 1$ **then** $\mathcal{D} \leftarrow 1/(1-p)$
else $\mathcal{D} \leftarrow +\infty$
end if
$root \leftarrow \mathcal{T}.root,\ root.d \leftarrow 1$
for all $ch \in root.children$ **do**
    $ch.d \leftarrow ch.degree,\ insert(ch, \Omega)$
end for
while $\Omega \neq \emptyset$ **do**
    $F \leftarrow delete(\Omega),\ NVroutes \leftarrow$ iSPFNV$(\mathcal{T}, F)$
    for all $Q \in F.chilren$ **do**
        $Q_F route \leftarrow$ get$(NVroutes, Q)$
        if $Q_F route.nexthop \neq Q.nexthop$ **then**
            $NotViaTable.add(Q_F route)$
        end if
        if $F.d \leq \mathcal{D}$ **then**
            $Q.d \leftarrow F.d \times Q.degree,\ insert(\Omega, Q)$
        end if
    end for
end while

Algorithm 1 is designed for R to find out all necessary NotVia addresses and compute routes for them. This algorithm traverses the SPT of R in the post order. When obtaining some node (F), the algorithm would take it as a failed node and compute the backup routes from R to all the children of F (T for example) if T_F is related to R according to p.

For each failed node F, one time of iSPFNV is required. Given R's SPT $\mathcal{T}$ and the failed node F, assuming that F fails, iSPFNV, which is revised from iSPF [17], computes the routes from R to all children of F.

IV. Performance Evaluation

In this section, we evaluate our ED-NotVia mechanism by analyzing the topologies collected by the Rocketfuel project [11] and generated by the Brite [12]. For the Rocketfuel data,

the topological characteristics are summarized in Table I. As the table shows that they have great diversity, we believe our evaluation can represent different topologies.

TABLE I
TOPOLOGICAL CHARACTERISTICS OF ROCKETFUEL DATA

AS Number	#PoP	#Router	#Link	Average degree
AS 1221	15	50	97	3.88
AS 3967	20	72	140	3.90
AS 3257	35	113	279	4.94
AS 6441	21	129	363	5.63
AS 1239	38	284	941	6.63
AS 7018	47	574	2021	7.04

A. Algorithm Validity

To examine the availability and performance of the Algorithm Nec-NotVia, we make a project modified from SSFNET to analyze the Rocketfuel and Brite data. The results demonstrate the benefits of our algorithm in several aspects:

- Averagely only a small proportion of NotVia addresses are actually necessary to some node R in the topologies, which is shown in Fig. 5. This is the ratio between the necessary and the SPT-related NotVia addresses of R, and the ratio between the necessary and all NotVia addresses would be much lower. Unnecessary NotVia addresses require no computing and memory, which confirms the value of our P-NotVia scheme.
- For the Rocketfuel topologies, the actual protection coverage is always not lower (mostly much higher) than *ratio-p* without exception, as shown in Fig. 8(a). Although this might not be strictly guaranteed for Brite topologies, it still works well, as is shown in Figure 9(a).
- The ratio of regarded-as-necessary (RaN) NotVia addresses is very low, especially for large ASs. Among these RaN NotVia addresses,there are some wrongly regarded-as-necessary address (W-RaN) addresses. Take As shown in Fig. 8(b), even if the *ratio-p* is set as 0.98, only about 10% of SPT-related NotVia addresses are regarded as necessary; and only 15% of these RsN NotVia addresses are not necessary actually. (At the same time, higher than 98% protection coverage is achieved).

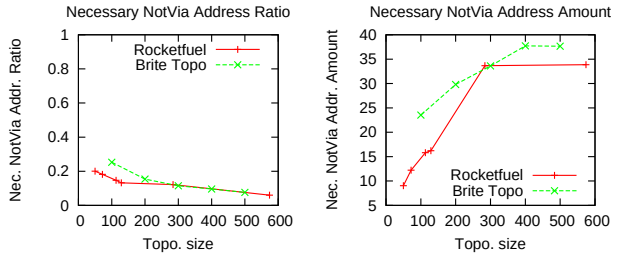


Fig. 5. Necessary NotVia addresses ratio and amount.

B. Protection Path Length Stretch

ED-NotVia is far better than NotVia on path length stretch (the ratio between the protection path and the optimal shortest path). Fig. 6 and Fig. 7 show the results from the topologies of both Rocketfuel and Brite.

As the earlier decapsulation is brought in, the protection path of ED-NotVia is much shorter than that of NotVia. For ED-NotVia, the protection path length stretch is either about 1.05 or much less than 1.05, which means the protection path of ED-NotVia is extremely close to the shortest path after the failure. This is a big improvement for NotVia.

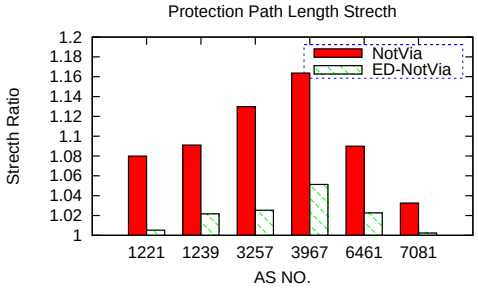


Fig. 6. Protection Path Length Strecth - Rocketfuel

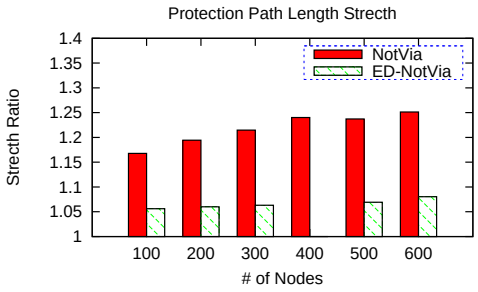


Fig. 7. Protection Path Length Strecth - Brite

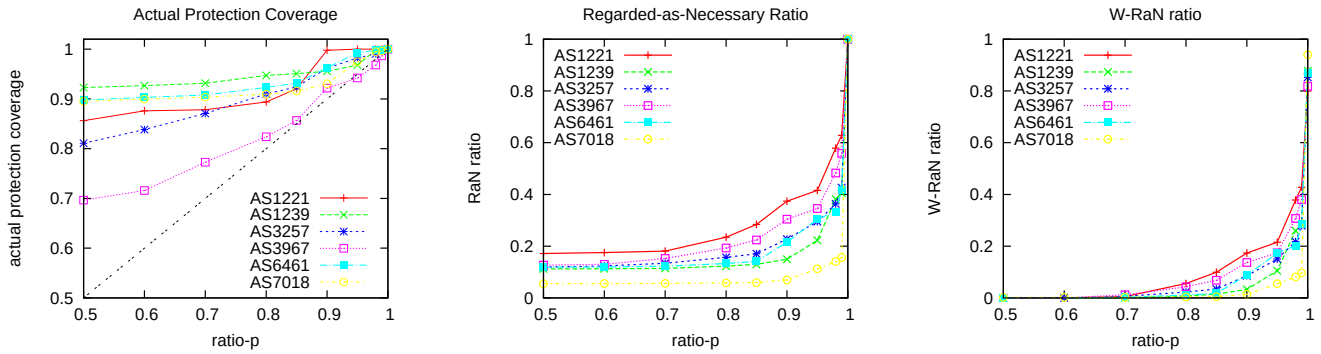
V. CONCLUSION AND FUTURE WORK

In this paper, we have presented ED-NotVia, which is a great improvement over NotVia. This mechanism provides higher protection coverage than LFA and is much more lightweight than Tunnel and NotVia. Compared with other improvements of NotVia ([6] and [14]), ED-NotVia is better in three aspects: first, it reduces NotVia addresses dramatically; second, it brings in the early decapsulation with marginal overhead, alleviating the path stretch of NotVia by 85%; third, we proposed a heuristic and flexible algorithm for a node to find necessary NotVia addresses and compute backup routes for them.

We believe that ED-NotVia is a practical approach to enhance the intra-domain network reliability. It can effectively reduce packet drops and network congestion during failure. However, ED-NotVia still does not support incremental deployment at the router level, just like NotVia. We are planning to work on this problem to make ED-NotVia more practical as the next step.

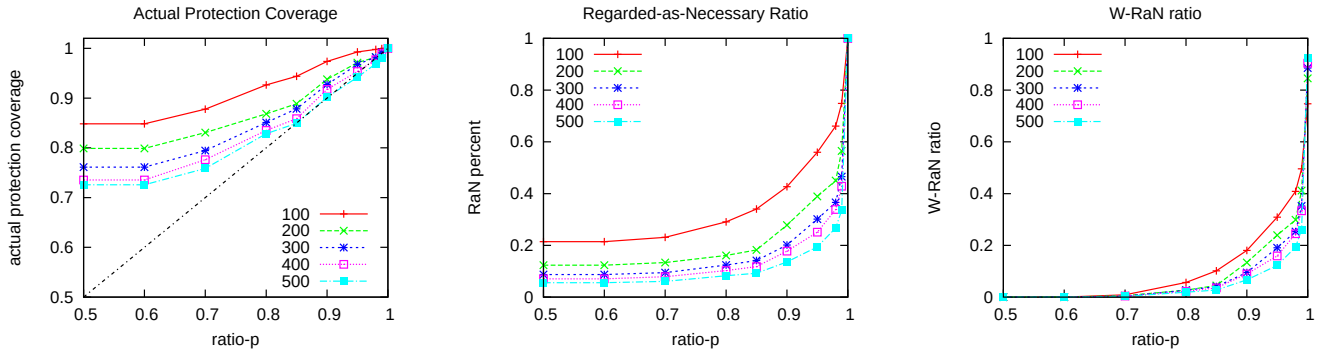
VI. ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (973) with No. 2009CB320502, the National High-tech R&D Program of China (863) with No. 2009AA01Z251, the National Science & Technology Pillar Program of China with No.2008BAH37B03, Hong Kong PolyU/GY-G78, A-PB0R, A-PJ19, 1-ZV5W, and RGC/GRF PolyU 5305/08E.



(a) Actual Protection Coverage. (b) Regarded-as-necessary (RaN) ratio. (c) Wrongly-RaN ratio.

Fig. 8. The verification of our algorithm based on Rocketfuel Topologies. (a) Actual protection coverage is always much higher than *ratio-p*, guaranteeing the validity of our algorithm. (b) RaN ratio is very low while compared with the protection coverage our algorithm achieves. (c) W-RaN ratio is less than 20% with 98% protection coverage, which means less computation is wasted.



(a) Actual Protection Coverage (b) RaN ratio (c) W-RaN ratio

Fig. 9. The verification of our algorithm based on Brite topologies, which respectively contain 100, 200, 300, 400 and 500 nodes but are with the same average degree of 6. The results can be explained in the same way as Fig. 8

REFERENCES

[1] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," in *Proc. SIGCOMM '00*, Stockholm, Sweden, Aug. 2000.

[2] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP restoration in a tier 1 backbone." *IEEE Network*, vol. 18, no. 2, pp. 13–19, 2004.

[3] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *SIGCOMM CCR*, vol. 35, no. 3, pp. 35–44, 2005.

[4] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving Convergence-free Routing Using Failure-carrying Packets," in *Proc. SIGCOMM'07*, Kyoto, Japan, Aug. 2007.

[5] M. Shand and S. Bryant, "IP Fast Reroute Framework," RFC 5714, Jan. 2010.

[6] A. Li, P. Francois, and X. Yang, "On Improving the Efficiency and Manageability of NotVia," in *Proc. CoNEXT'07*, New York, NY, Dec. 2007.

[7] A. Raj and O. C. Ibe, "A Survey of IP and Multiprotocol Label Switching Fast Reroute Schemes," *Comput. Netw.*, vol. 51, no. 8, pp. 1882–1907, 2007.

[8] A. Atlas and A. Zinin, "Basic Specification for IP Fast-Reroute: Loop-free Alternates," RFC 5286, Sep. 2008.

[9] S. Bryant, C. Filsfils, S. Previdi, and M. Shands, "IP Fast Reroute Using Tunnels," Internet Draft, draft-bryant-ipfrr-tunnels-03, Nov. 2007.

[10] M. Shands, S. Bryant, and S. Previdi, "IP Fast Reroute Using Not-via Addresses," Internet Draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-05, Mar 2010.

[11] Rocketfuel: An ISP Topology Mapping Engine, http://www.cs.washington.edu/research/networking/rocketfuel.

[12] BRITE Project, http://www.cs.bu.edu/brite/index.html.

[13] A. Atlas, R. Torvi, G. Choudhury, and D. Fedyk, "Ospfv2 extensions for link capabilities to support u-turn alternates for IP/LDP fast-reroute," Internet Draft, draft-atlas-ospf-local-protect-cap-02, Feb. 2006.

[14] G. Enyedi, G. Rétvári, P. Szilágyi, and A. Császár, "IP Fast ReRoute: Lightweight Not-Via," in *Proc. NETWORKING'09*, Aachen, Germany, May 2009.

[15] T. Cicic, A. F. Hansen, and O. K. Apeland, "Redundant Trees for Fast IP Recovery," in *Proc. BROADNETS'07*, Raleigh, NC, Sep. 2007.

[16] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of Failures in an IP Backbone," in *Proc. IEEE INFOCOM'04*, Mar. 2004.

[17] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic spt algorithm based on a ball-and-string model," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 706–718, 2001.