ARTICLE IN PRESS

Computer Networks xxx (2011) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time ☆

Mingwei Xu, Yuan Yang *, Qi Li

Department of Computer Science & Technology, Tsinghua University, China

ARTICLE INFO

Article history:
Received 26 May 2011
Received in revised form 26 September 2011
Accepted 11 November 2011
Available online xxxx

Keywords:
IP routing
Fast rerouting
Tunnel
Fast Tunnel Selection

ABSTRACT

IP Fast ReRoute (IPFRR) has received increasing attention as a means to effectively shorten traffic disruption under failures. A major approach to implementing IPFRR is to pre-calculate backup paths for nodes and links. However, it may not be easy to deploy such an approach in practice due to the tremendous computational overhead. Thus, a light-weight IPFRR scheme is desired to effectively provide cost-efficient routing protection. In this paper, we propose a Fast Tunnel Selection (FTS) approach to achieve tunnel-based IPFRR. FTS approach can find an effective tunnel endpoint before complete computation of entire SPT and thus effectively reduce computation overhead. Specially, we propose two FTS algorithms to provide protection for networks with symmetric and asymmetric link weights. We simulate FTS with topologies of different sizes. The results show that FTS approach reduces more than 89% computation overhead compared to the existing approaches, and achieves more than 99% average link protection rate and more than 90% average node protection rate. Moreover, FTS approach achieves less than 15% path stretch, which is better than the existing approaches.

1. Introduction

Fast evolution of network applications requires high availability and stability of Internet routing. However, the Internet routing is not well resilient to failures. A link or node failure will trigger routing convergence during which routes are rebuilt among routers, and traffic may be disrupted during this period. Generally, link state routing protocols require several seconds for convergence. Thus, they cannot provide immediate connectivity to all routers after failures [2]. To address this issue, IP resilience to ameliorate fault recovery is gaining increasing attention.

IP Fast ReRoute (IPFRR) [3] provides a potential technique to improve IP resilience in intra-domain routing. IPFRR switches traffic to backup routes quickly after failures occur, and greatly shorten the interruption period to tens of microseconds with the help of fast failure detecting techniques [4]. In general, IPFRR approaches can be implemented by different backup path selection algorithms, most of which need to consume lots of router resources, e.g., CPU cycles, and may exacerbate router performance. Thus, computation overhead introduced by existing IPFRR schemes is significant.

To reduce the computation overhead, several improved tunnel-based IPFRR solutions are proposed. Li et al. reduced the number of shortest path tree (SPT) computation with the Notvia approach in IPFRR [5] so that a router only needs to calculate a few SPTs. Enyedi et al. improved

☆ This journal version is an extension of our conference paper "A Lightweight IP Fast Reroute Algorithm with Tunneling", which is published in the proceedings of international conference on communication (ICC) [1]. This journal version makes the following extensions to the conference paper: we address rerouting path selections in networks with symmetric link weights to improve the protection efficiency; we prove that the overall computational complexity of rerouting path selection algorithm with symmetric link weights is linear with respect to the number of nodes and links; we evaluate our proposed algorithms using large-scale real topologies to demonstrate the performance of our scheme in production networks, and discuss the application scenarios of different IPFRR approaches.

* Corresponding author. Tel./fax: +86 10 6278 5822.
E-mail address: yyang@csnet1.cs.tsinghua.edu.cn (Y. Yang).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.11.006

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

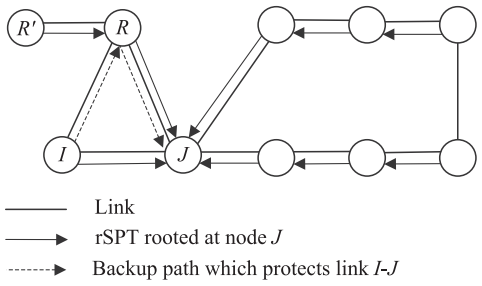


Fig. 1. A sub-set of CERNET2 [7] topology where solid lines denote links, dashed arrows denote a backup path from node *I* to node *J*, and solid arrows denote the reverse shortest path tree (rSPT) rooted at node *J*, which is consisted of all the shortest paths to node *J*. Node *I* will shift traffic to node *R* in order to enable the backup path in case that link $I \rightarrow J$ fails.

the Notvia approach by applying redundant trees [6]. The computing time is reduced by an order of magnitude, and the time is only in the range of hundreds of milliseconds. Unfortunately, the overhead of these proposals is still large because they need to compute the entire SPT whose computational complexity is $O(|E| + |V|\log|V|)$, where E is the set of links and V is the set of nodes. Moreover, existing IPFRR does not well consider the path stretch issue. That is, IPFRR may result in large end-to-end delay and bandwidth waste.

Example: Let us use a topology shown in Fig. 1 as an example to understand the computation overhead and the path stretch problem in traditional IPFRR schemes. In traditional IPFRR, backup routing paths are found by computing reverse shortest path tree (rSPT) rooted at each node. rSPT is consisted of all the shortest paths to a destination node and is also called a sink tree. For example, node *I* computes the rSPT rooted at node *J* to protect link $I \rightarrow J$, and the number of operations in this computation is about 32[1] using the Dijkstra algorithm. Normally, it is not necessary to compute the entire rSPT. Instead, we only compute the reverse shortest path from node *J* to its neighbor *R* and check if it can provide a backup path, and then the number of operations is only 12.5. The example is not special. In general, we only need to compute a small part of rSPTs. Furthermore, the path stretch introduced by traditional IPFRR schemes is not always optimal. For example, the protection path via node *R*′ is longer than that via node *R*, however, existing approaches may not select node *R* as the protection endpoint since they did not consider decreasing path stretches during backup path selections.

In this paper, we propose a light-weight backup path selection approach to achieve efficient tunnel-based IPFRR. The approach introduces little computation overhead and the rerouting paths only incur small path stretches. We first propose a Symmetric Fast Tunnel Selection (SymFTS) algorithm to select rerouting paths in networks with symmetric link weights. In SymFTS, we select a tunnel endpoint according to traditional routing information in Interior Gateway Protocol (IGP), i.e., Link State Data Base (LSDB) and the calculated SPT. The algorithm has a linear computational complexity in each node, i.e. $O(2|E| + |V|)$. In particular, in topologies where the links are sparse, the saving will be more significant, and the path stretches incurred will be much smaller.

Furthermore, we propose an improved algorithm called Asymmetric Fast Tunnel Selection (AsymFTS) to improve protection effectiveness of networks with asymmetric link weights. The algorithm marks nodes that can be reached without traversing a component under protection, e.g., link or node. A marked node will be chosen as the tunnel endpoint immediately once we determine that the shortest path from the marked node to the node at the other side of the component does not traverse the component. In this way, the AsymFTS algorithm finds an effective tunnel endpoint before the end of the entire SPT computation, and thus effectively reduces computation overhead. The AsymFTS algorithm introduces small path stretches as well as the SymFTS algorithm.

We evaluate our algorithms by simulation. The simulation results show that our algorithms decrease the computation overhead by at least 89% compared to existing approaches, such as LFA [8], Uturns [9], Tunnels [10] and Notvia [11]. In particular, our proposed algorithms provide more than 99% average link protection rate and more than 90% average node protection rate, which is much better than LFA and close to Uturns and Notvia. Moreover, our algorithms achieve less than 15% path stretch, which is better than the existing approaches.

The remainder of the paper is organized as follows. In Section 2, we present the background and related work. Our algorithms are proposed in Sections 3 and 4. We evaluate our algorithm in Section 5. Section 6 discusses the application scenarios of different IPFRR approaches. Section 7 concludes the paper.

2. Background and related work

Reactive and proactive approaches are two major techniques for IP resilience [12]. Reactive approaches aim at shortening the convergence time of routing protocols, and proactive approaches, which are also called protection approaches, aim at reducing packet-loss by rerouting packets to backup paths.

Reactive approaches can deal well with multiple link failures. A typical approach is to shorten the convergence time for IGP [2]. However, if failed links recover faster than routing convergence, routing flap may occur. Proactive approaches, such as IPFRR, can handle single link or node failure within a very short time. They do not advertise failure information when failures are detected, and provide network connectivity by rerouting the traffic to the backup paths. Since failure detection and backup path activation only last a short time, these approaches can protect against failures quickly.

Several approaches are proposed to implement IPFRR, such as Loop Free Alternates (LFA) [8], Uturns [9], Tunnels [10,13], and Notvia addresses [11]. All these approaches use tunnels or directed forwarding to deliver packets in backup paths. The LFA approach utilizes neighbors'

[1] The number of operations may differ with different implementations of the Dijkstra algorithm, but the trends are similar.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

shortest paths to build protection paths. Thus, extra SPTs rooted at the neighbors must be computed. The Uturns approach considers neighbors' neighbors as alternate nodes to forward traffic, and extra SPTs rooted at the nodes within two hops must be computed. Similarly, the Tunnels approach also uses alternate nodes called tunnel endpoints to build protection paths. The Tunnels approach also needs several extra SPT computations. The Notvia approach uses Notvia addresses to indicate failure identities and distributes these Notvia addresses in the network. Packets encapsulated with the addresses are steered round failures. Thus, an extra SPT computation is required for each Notvia address. To improve the efficiency of IPFRR, combination of LFA and Notvia is also proposed. However, Menth et al. found that the combined usage of both methods has no advantage compared with applying Notvia addresses only [14]. These approaches all require computing several extra entire SPTs [15].

2.1. Traditional Tunnels approach

In the traditional Tunnels approach, node I is supposed to protect the links connecting with its neighbor nodes. In order to find protection paths, target nodes which potentially can provide protection for failed links or nodes must be identified first. *Generally, all neighbors of node I are treated as the targets to protect links, and nodes who are the two-hop neighbors of node I are treated as the targets to protect nodes.*[2] After identifying targets, a tunnel endpoint must be chosen for each target. A tunnel endpoint N is a node to which node I sends encapsulated packets when it detects a failure between nodes I and J, and it decapsulates the packets and enables normal packet forwarding.

Node I must insure that the packets can reach their destinations by detouring the failure, i.e. without looping back to node I. Let us assume that we need to protect link $I \to J$. Node N would be an effective tunnel endpoint when the following conditions hold:

$$J \notin \mathrm{SP}(I,N), \tag{1}$$

$$I \notin \mathrm{SP}(N,J), \tag{2}$$

where $\mathrm{SP}(I,N)$ indicates the shortest path from node I to node N. If we compute the cost of packet forwarding with the shortest path, Eqs. (1) and (2) can be rewritten as

$$\mathrm{cost}(I,N) < \mathrm{cost}(I,J) + \mathrm{cost}(J,N), \tag{3}$$

$$\mathrm{cost}(N,J) < \mathrm{cost}(N,I) + \mathrm{cost}(I,J). \tag{4}$$

Now let us briefly review how a tunnel endpoint is selected in the traditional Tunnels mechanism [10].

Step 1. A set of nodes which can be reached from node I by normal forwarding without traversing link $I \to J$ is calculated. It is termed as Pspace of node I with respect to link $I \to J$. The Pspace can be obtained by SPT(I) with all nodes that are reached via link $I \to J$ pruned.

Step 2. A set of nodes which can reach node J without traversing link $I \to J$ is calculated. This set is termed as Qspace of node J with respect to link $I \to J$. Qspace can be obtained by computing a reverse shortest path tree (rSPT) rooted at node J and pruning the nodes that reach node J via link $I \to J$.

Step 3. The set of candidate tunnel endpoints candidates(J) is the intersection between Pspace and Qspace. We can directly select the tunnel endpoint from the set candidates(J).

To provide a node protection instead of link protection, the procedure is similar to above, therefore we do not repeat it here. It is obvious that the Tunnels approach consume a lot of CPU cycles for computing rSPT(J). Meanwhile, the traditional approach does not discuss how to select a tunnel endpoint from candidates(J). A poor selection of tunnel endpoint may incur large path stretch.

2.2. Related works

Lots of studies improved tunnel-based IPFRR solutions to reduce the computation overhead. Li et al. [5] found that protection paths do not traverse many nodes and thus these nodes do not need to compute extra SPTs for these paths. However, a node still needs to compute many paths which traverse it, especially when failures are connected to the node. Ho et al. [13] proposed a tunnel endpoint selection algorithm so that the network performance is optimized after the repaired paths are activated for rerouting. However, the complexity is high and even higher than traditional Tunnels approach. Enyedi et al. [6] decreased the number of required Notvia addresses by reformulating rerouting path computations in terms of redundant trees. Each node computes a single pair of redundant trees and the overall computational complexity is $O(|E| + |V|)$. However, the address management and path stretch issues are not well addressed. In summary, these improved IPFRR schemes fail to effectively reduce the overhead since they did not consider reducing several extra SPT computations.

Moreover, Kini et al. [16] proposed a tunnel-based proposal which provides resilience for up to two link failures. The approach requires three protection addresses for every node besides the normal address. Every protection address is associated with a protection graph, and thus it brings large management and computation overhead. Nelakuditi et al. [17] proposed a local rerouting approach called failure insensitive routing. The approach prepares rerouting paths under failures using interface-specific forwarding. They computed forwarding and backwarding tables with the available shortest path first (ASPF) algorithm which computes the shortest paths excluding the unavailable (potentially failed) links. The complexity of ASPF is $O(|E| \times \log^2(|V|))$. Ray et al. [18] presented a backup path selection algorithm named Distributed Path Computation with Intermediate Variables (DIV) which guarantees that the directed graph (i.e., routing paths) induced by the routing decisions remains acyclic at all times. DIV uses a non-shortest path routing and thus increases path stretch. Francois et al. [19] proposed an approach to progressively change link weights to ensure stable and loop-free routing

[2] In this paper, for simplicity but without loss of generality, we assume that networks under protection do not contain equal cost multi-path.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

path changes. However, each link weight change requires a new SPT computation. Kvalbein et al. [20] proposed an approach to protect against single failure by using multiple routing configuration. Each link is isolated in at least one configuration to provide protection. Kwong et al. [21] proposed protection routing to deliver packets without tunneling when a failure occurs. Similar to the traditional IPFRR, these schemes did not consider reducing computation overhead and path stretch.

3. Fast Tunnel Selection for symmetric link weights

In networks with symmetric link weights, it is well known that cost(I,J) = cost(J,I) for any node pair I and J. Based on this property, we can obtain some useful theorems for fast tunnel endpoint selection by Graph Theory. Link and node protections can be considered separately because we have different considerations of tunnel endpoint selection though the basic idea is similar and achieves linear computational complexity.

The notations used in this paper are summarized in Table 1.

3.1. Link protection

In order to protect link $I \to J$, we set node J as the target node just like in traditional Tunnels approach. However, we use the conception of subtree instead of Pspace and Qspace. subtree (I,J) is the node set whose reverse shortest paths to node I traverse node J, where J is a neighbor of I. According to the definitions of Pspace, Qspace and subtree, we obtain

$$V - \text{Pspace}(I, I \to J) = \text{subtree}(I,J), \text{ and}$$
$$V - \text{Qspace}(J, I \to J) = \text{subtree}(J,I).$$

Due to the definition of candidates(J), we obtain

$$\text{candidates}(J) = V - \text{subtree}(J,I) - \text{subtree}(I,J). \quad (5)$$

The following theorem shows the candidate node space for tunnel endpoint selections, and we can effectively limit the search range to reduce computation overhead.

Theorem 3.1. *In a network with symmetric link weights, there is a link $R \to R'$ s.t. $R \in$ subtree(I,J) and $R' \in$ candidates(J), if $I \to J \in$ SP(I,J) and candidates(J) $\neq \Phi$.*

Proof. The proof is by contradiction. Suppose there is not any link $R \to R'$ s.t. $R \in$ subtree(I,J) and $R' \in$ candidates(J). Thus for any node R' in candidates(J), the links originated from node R' only connect to nodes in $V -$ subtree(I,J). Due to Eq. (5), $V -$ subtree(I,J) equals candidates(J) + subtree(J,I), because the intersection of subtree(J,I) and subtree(I,J) is empty. Therefore, SP(R',J) must traverse some nodes in subtree(J,I). Due to the symmetry of link weights, the shortest path from the nodes in subtree(J,I) to node J must traverse node I, so $I \in$ SP(R',J). This contradicts to $R' \in$ candidates(J). □

Fig. 2 illustrates an example explaining Theorem 3.1. We can get subtree(I,J) (the gray nodes shown in Fig. 2) by computing SPT(I). Theorem 3.1 gives a simple way to find possible tunnel endpoints: traverse subtree(I,J) and check each nodes' neighbors which are not included in subtree(I,J). We have the following theorem to identify a tunnel endpoint from the node space obtained by Theorem 3.1.

Theorem 3.2. *In a network with symmetric link weights, $R' \in$ candidates(J) if the following conditions hold: (1) $I \to J \in$ SP(I,J); (2) There is a link $R \to R'$ s.t. $R \in$ subtree(I,J) and $R' \notin$ subtree(I,J) and $R' \neq I$; and (3)*

$$2 \cdot \text{cost}(I,J) > \text{cost}(I,R) - \text{cost}(I,R') + \text{cost}(R,R'). \quad (6)$$

Proof. The proof is generally obtained by deducing from Inequation (6).

According to the fact that cost(I,J) is equal to cost(J,I) (which always holds under a network connected with symmetric link weights) and Inequation (6), we obtain

$$\text{cost}(J,I) + \text{cost}(I,R') > \text{cost}(I,R) + \text{cost}(R,R') - \text{cost}(I,J). \quad (7)$$

According to that $R \in$ subtree(I,J), we obtain

$$\text{cost}(I,R) = \text{cost}(I,J) + \text{cost}(J,R). \quad (8)$$

We can rewrite Eq. (7) with the value of cost(I,R) in Eq. (8), and obtain

$$\text{cost}(J,I) + \text{cost}(I,R') > \text{cost}(I,J) + \text{cost}(J,R) + \text{cost}(R,R') - \text{cost}(I,J).$$

Table 1
Summary on notations.

Notation	Meaning
$Graph(V,E)$	The network topology with vertex (node) set V and edge (link) set E
I,J,K,N,R,R',R'',R_i,R_j	Nodes in V. I often denotes the node connected to a failure and J often denotes the target node
$I \to J$	A directed link from node I to node J
SP(I,J)	Shortest path from node I to node J
SPT(I)	Shortest path tree rooted at node I
rSPT(I)	Reverse shortest path tree rooted at node I
cost(I,J)	Cost of shortest path from node I to node J
candidates$J$	The set of candidate tunnel endpoints for target node J
Pspace (I,$I \to J$)	The set of nodes that node I reaches using normal forwarding without traversing link $I \to J$
Pspace (I,K)	The set of nodes that node I reaches using normal forwarding without traversing node K
Qspace (J,$I \to J$)	The set of nodes that can reach node J using normal forwarding without traversing link $I \to J$
Qspace (J,K)	The set of nodes that can reach node J using normal forwarding without traversing node K
subtree(I,J)	SPT(I)'s subtree which is rooted at node J

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

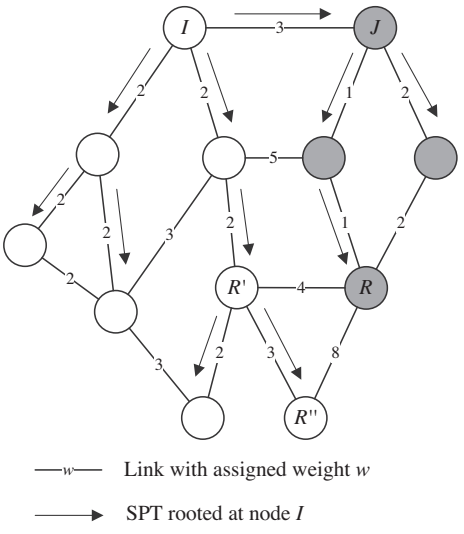


Fig. 2. An example network topology. Grey nodes are in subtree(I,J). Node R' and node R'' are neighbors of node R and are in candidates(J).

Thus, $\text{cost}(R',I) + \text{cost}(I,J) > \text{cost}(J,R) + \text{cost}(R,R') \geqslant \text{cost}(J,R')$

As $\text{cost}(J,R')$ is equal to $\text{cost}(R',J)$, we obtain

$$\text{cost}(R',I) + \text{cost}(I,J) > \text{cost}(R',J). \quad (9)$$

Moreover, since $R' \notin \text{subtree}(I,J)$, we obtain

$$\text{cost}(I,R') < \text{cost}(I,J) + \text{cost}(J,R'). \quad (10)$$

According to Inequations (9) and (10), we can obtain $R' \in \text{candidates}(J)$. □

The key condition in Theorem 3.2 is Inequation (6). The values in Inequation (6) are directly obtained from Link State Data Base (LSDB), i.e. $\text{cost}(I,J)$ and $\text{cost}(R,R')$, and from computing SPT(I), i.e. $\text{cost}(I,R)$ and $\text{cost}(I,R')$. Since SPT(I) is calculated and stored in the traditional IGP protocols, we can directly identify whether node R' is a tunnel endpoint.

Note that, since Inequation (6) is a sufficient but not necessary condition, the Inequation may not hold if node R' is a tunnel endpoint. The example in Fig. 2 illustrates the case, where R'' is a tunnel endpoint, but

$$2 \cdot \text{cost}(I,J) = \text{cost}(I,R) - \text{cost}(I,R'') + \text{cost}(R,R'') = 6.$$

In this example, packets sent by I to J will traverse R' and $R' \to R''$ twice if we use R'' as the tunnel endpoint after link $I \to J$ fails. This is a path-overlapping problem which causes unnecessary path stretches. Fortunately, we can solve the problem by selecting a tunnel endpoint which provides the shortest protection path. Here, the length of protection paths can be calculated quickly by:

$$\begin{aligned}\text{cost}(I,R') + \text{cost}(R',J) &= \text{cost}(I,R') + \text{cost}(R,R') + \text{cost}(R,J) \\ &= \text{cost}(I,R') + \text{cost}(R,R') + \text{cost}(I,R) \\ &\quad - \text{cost}(I,J). \end{aligned} \quad (11)$$

We can prove that selecting shortest protection path never raises path-overlapping (see Lemma 3.2).

Based on the discussions above, we select tunnel endpoints for a network with symmetric link weights as follows. Firstly, we need to traverse the nodes on subtree(I,J). For every node R in subtree(I,J), we evaluate if its neighbor R' is neither node I nor in subtree(I,J). Then, we can find the R' which satisfies Inequation (6) and has the shortest protection path. The pseudo-code is shown in Algorithm 1. The input of SymFTS are SPT(I),J and *Graph*(V,E). The algorithm will return a node once the node is chosen as the tunnel endpoint, otherwise return null.

Lemma 3.1. *In a network with symmetric link weights, the computation overhead in each node is $O(2|E| + |V|)$ using the SymFTS algorithm.*

Algorithm 1. The SymFTS Algorithm

Input: SPT(I), J, *Graph*(V,E);
Output: the tunnel endpoint;
1: mark all the nodes reached from node I via link $I \to J$ with red, making use of SPT(I);
2: mark node I with red;
3: *tunnel_endpoint* ⇐ null;
4: *mincost* ⇐ infinity;
5: **for** each node R in subtree(I,J)
6: **for** each node R' which has a link to R
7: **if** R' is red **or**
$2 \cdot \text{cost}(I,J) < \text{cost}(I,R) - \text{cost}(I,R') + \text{cost}(R,R')$
or
$\text{cost}(I,R') + \text{cost}(R,R') + \text{cost}(I,R) - \text{cost}(I,J) \geqslant \textit{mincost}$
8: continue;
9: *tunnel_endpoint* ⇐ R';
10: *mincost* ⇐ $\text{cost}(I,R') + \text{cost}(R,R') + \text{cost}(I,R) - \text{cost}(I,J)$;
11: **return** *tunnel_endpoint*;

The proof of Lemma 3.1 is trivial. We only show the proof outline here. To protect the m links connected to I, we need to run the SymFTS algorithm m times. Note that each node except node I will only be visited once in the algorithm (see steps 5–10) and each link will be visited twice (see steps 6–10). Thus, the computational overhead of a node is $O(2|E| + |V|)$, which is a linear complexity. Compared to traditional Tunnels approach whose computational overhead is $O(m|E| + m|V|\log|V|)$ at least, SymFTS achieves much low complexity.

Lemma 3.2. *In a network with symmetric link weights, R' is a tunnel endpoint selected by SymFTS, and then there does not exist node R'' s.t. $R'' \neq R'$ and $R'' \in \text{SP}(I,R')$ and $R'' \in \text{SP}(R',J)$.*

The proof of Lemma 3.2 can be found in Appendix. Lemma 3.2 states that there does not exist any node in both SP(I,R') and SP(R',J) with SymFTS. Therefore, SymFTS ensures that packets will never traverse a link or a node more than once when a failure occurs. It effectively reduces end-to-end delays and saves bandwidth as well as process cost, which is benefit from a small path stretch.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

3.2. Node protection

In the situation of protecting a neighbor node K, we set the target node to a two-hop neighbor J which is on the shortest path to the destination. Similar to Eq. (5), we obtain

$$\text{candidates}(J) = V - \text{subtree}(J,K) - \text{subtree}(I,K).$$

We can obtain the following Theorems which are similar to Theorems 3.1 and 3.2.

Theorem 3.3. *In a network with symmetric link weights, there is a link $R \to R'$ s.t. $R \in$ subtree(I,K) and $R' \in$ candidates(J), if $I \to J \in$ SP(I,K) and $J \to K \in$ SP(I,K) and candidates(J) $\neq \Phi$.*

Theorem 3.4. *In a network with symmetric link weights, $R' \in$ candidates(J) if the following conditions hold: (1) $I \to J \in$ SP(I,K) and $J \to J \in$ SP(I,K); (2) There is a link $R \to R'$ s.t. $R \in$ subtree(I,K) and $R' \notin$ subtree(I,J) and $R' \neq I$; (3) $2 \cdot$ cost(I,J) > cost(I,R) − cost(I,R′) + cost(R,R′).*

Theorems 3.3 and 3.4 states the conditions of node space and endpoint selection for node protection, respectively. Since the proofs of Theorems 3.3 and 3.4 are similar to those of Theorems 3.1 and 3.2, we do not repeat it here. According to Theorem 3.4, we can easily select tunnel endpoints for node protection by simply extending SymFTS (see Algorithm 1). That is, we can change link $I \to J$ to link $I \to K$ in step 1. The computational complexity remains at $O(2|E| + |V|)$. Note that the algorithm for node protection will ignore some candidate tunnel endpoints, as we may overlook some nodes in subtree(I,K) − subtree(I,J) in the algorithm. As a result, the protection rate may reduce a little. However, the results are still acceptable (see the simulation results in Section 5). Here, the protection rate indicates how many links or nodes can be protected, considering if the component (i.e., link or node) is used by a routing path. The formal definition of the protection rate can be found in Section 5.3.

Protection for asymmetric link weight networks is more complicated. If we directly apply the SymFTS algorithm in these networks, routing loops may incur under failures. We will propose an improved algorithm called AsymFTS to address this issue in the following section.

4. Fast Tunnel Selection for asymmetric link weights

In this section, we propose the AsymFTS algorithm to address tunnel endpoint selection for networks with asymmetric link weights. Different from that in the networks with symmetric link weights, we can only know the path costs from I using SPT(I) in the networks with asymmetric link weights. Thus, the reverse shortest path from target node J (rSPT(J)) should be calculated in order to guarantee that the shortest path from the tunnel endpoint to J does not traverse the failure component.

However, we believe that it is not necessary to calculate the entire rSPT because we can mostly identify an effective tunnel endpoint before finishing rSPT computations. Our approach is to combine the tunnel endpoint selection process with the rSPT computation. In this way, rSPT computation can be terminated as soon as we identify a tunnel endpoint. Note that the difference between link protection and node protection is much less than that in networks with symmetric link weights, and thus we can consider them together here. Our goal is to find a tunnel endpoint if there exists any, either link or node protection.

When there exist several candidate tunnel endpoints, we select the nearest one to the target node J as the tunnel endpoint. Normally, the nearest tunnel endpoint to J can be identified before computation of rSPT(J) finishes. The algorithm called AsymFTS is shown in Algorithm 2. The input of AsymFTS are SPT(I), J and $Graph(V,E)$ (K is also needed for node protection). The output is the selected tunnel endpoint.

In AsymFTS, we calculate reverse shortest path from J by using the Dijkstra algorithm. We check whether node R_i meets the conditions, i.e., Eqs. (3) and (4), to be a tunnel endpoint once the reverse shortest path from J to R_i is determined. If R_i meets the condition, it will be selected as the tunnel endpoint and the algorithm stops. Otherwise, we should start a new round of reverse shortest path calculation and tunnel endpoint selection. AsymFTS can leverage the results obtained by each round of reverse shortest path calculation to identify the smallest-cost node, and then find the nearest tunnel endpoint to node J as early as possible.

As we discussed, SymFTS has a computation complexity of $O(2|E| + |V|)$. However, AsymFTS leverages the Dijkstra algorithm to search potential endpoint and will stop searching once a tunnel endpoint is found. Therefore, AsymFTS will spend more time to search endpoint than SymFTS in the networks with low link densities. In the worst case, when a network has a ring topology, the complexity will be $O(|E| + |V|\log|V|)$ and rSPT(J) needs to be computed. Fortunately, we need much less computation in most cases, especially in networks connected by a lot of links. Generally, the computation overhead of AsymFTS is similar to that of SymFTS in the networks with high link densities. The following Lemma states the distribution of tunnel endpoint if there exist more than one endpoint. The proof of Lemma 4.1 can be found in Appendix.

Lemma 4.1. *Node R'' is in candidates(J) if node R' is in candidates(J), where R'' is downstream of R' in rSPT(J).*

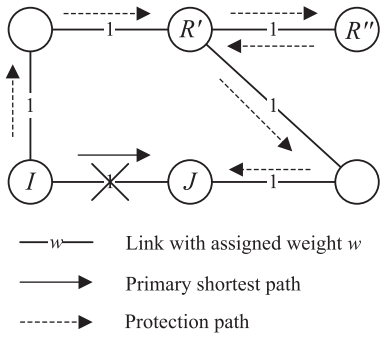


Fig. 3. Protecting link $I \to J$ using R'' as a tunnel endpoint.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

Algorithm 2. The AsymFTS algorithm

Input: SPT(I), J, $Graph(V,E)$, K for node protection;
Output: the tunnel endpoint;
1: mark all the nodes in subtree(I,J) (or subtree(I,K) for node protection) with blue, making use of $SPT(I)$;
2: **for** each node R_i in $Graph(V,E)$
3: $distance[R_i] \Leftarrow$ infinity;
4: $previous[R_i] \Leftarrow$ undefined;
5: $distance[J] \Leftarrow 0$;
6: $Q \Leftarrow V$;
7: **while** Q is not empty
8: $R_i \Leftarrow$ node in Q with the smallest $distance[\,]$;
9: remove R_i from Q;
10: **if** $previous[R_i] = I$ (or K) **or** $previous[R_i]$ is red
11: mark R_i with red;
12: **if** R_i is not blue and not red
13: **return** R_i;
14: **for** each link $R_j \rightarrow R_i$ ingoing to R_i
15: **if** $distance[R_j] > distance[R_i] + \text{cost}(R_j,R_i)$
16: $distance[R_j] \Leftarrow distance[R_i] + \text{cost}(R_j,R_i)$;
17: $previous[R_j] \Leftarrow R_i$;
18: **return** null;

Lemma 4.1 does not state whether a tunnel endpoint is near to node J but state the candidate nodes in the subtree rooted at a tunnel endpoint. The number of candidate nodes that hide in the subtree will increase quickly with the increase of network sizes. In other words, they do not need to be determined in AsymFTS and thus we can save lots of CPU cycles.

The following lemma states that AsymFTS solves the path-overlapping problem. The proof of the lemma can also be found in Appendix.

Lemma 4.2. *Node R' is the tunnel endpoint which is the nearest to target node J, then there is no node R'' s.t. $R'' \neq R'$ and $R'' \in SP(I,R')$ and $R'' \in SP(R',J)$.*

Let us take an example in Fig. 3 where packets are forwarded over both two directions of a link. Link costs are all set to 1 and R'' can be used as a tunnel endpoint to protect link $I \rightarrow J$. Packets will traverse link $R' \rightarrow R''$ and link $R'' \rightarrow R'$, and traverse node R' twice, which causes unnecessary path stretches. To address this issue, AsymFTS will directly select R' as the tunnel endpoint to protect link $I \rightarrow J$. Although protection paths calculated by AsymFTS may not be the shortest, packets will not traverse any node more than once.

5. Performance evaluation

In this section, we evaluate our proposed algorithms by simulation, compared with existing IPFRR approaches, such as LFA, Uturns, Tunnels and Notvia. We firstly present methodology for simulating different proposals and then show simulation results.

We implemented a simulator to implement different IPFRR proposals, such as LFA, Uturns, traditional Tunnels, Notvia, and SymFTS and AsymFTS. These proposals have different strategies to choose alternate nodes when there are several candidate tunnel endpoints. LFA and Uturns use random alternate nodes, and traditional Tunnels uses the nearest tunnel endpoint to node I. We use link/node protection in our simulation and evaluate end-to-end connectivity under each link/node failure. We use both generated and real topologies in our simulations.

Firstly, we use two real topologies of Abilene [22] and CERNET2 [7], to evaluate the performance of different IPFRR scheme. Secondly, we use 6 real topologies obtained from rocketfuel [23]. Since the networks in AS 1221 and AS 6461 in [23] are not completely connected and these ASes consist of different disconnected subnetworks, we only choose the largest subnetworks in these two ASes and drop the small ones which include at most 4 nodes. Since the link weights of publicly available network topologies are symmetric, for simplicity without loss of generality, we only evaluate both SymFTS and AsymFTS in networks with symmetric link weights.

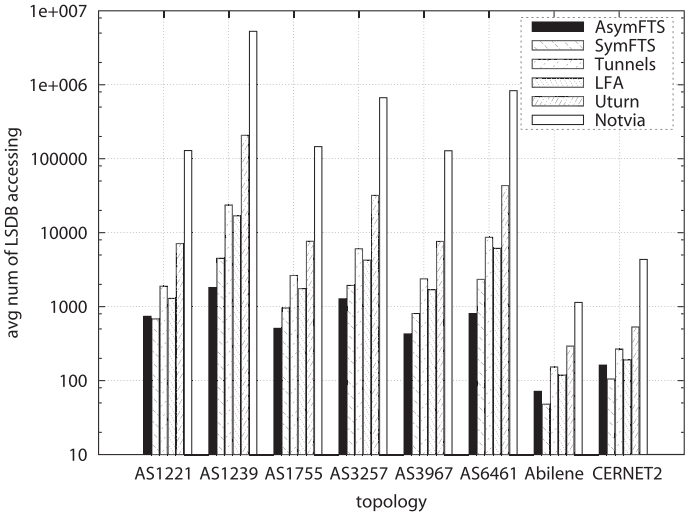


Fig. 4. Computation overhead with link protection in real topologies.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

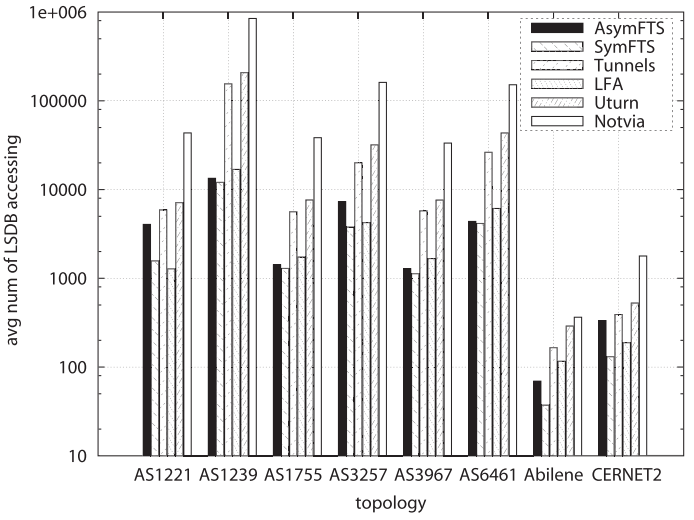


Fig. 5. Computation overhead with node protection in real topologies.

Furthermore, we use BRITE [24] to generate some large scale topologies. The parameters required by BRITE are based on the studies in [25], where the type is Bottom up and the router model is RTGLP. The average number of links per new node is two. The node place follows a heavy-tail distribution and the link costs are based on link latencies. The number of nodes varies from 10 to 200. We generate 10 different topologies for each size, and the results are average values of them.

5.1. Computation overhead

Firstly we evaluate computation overhead of different proposals. Generally, the basic operations during execution of the Dijkstra algorithm are Link State Database (LSDB) accessing, cost comparison and computation. Every LSDB accessing is a process to search and access Link State Advertisements (LSA) announced by all routers. LSDB accessing consumes the majority of CPU cycles and requires much more CPU cycles than the operation of cost comparison and computation. Thus, in this experiment, we evaluate the computation overhead by measuring the number of LSDB accessing.

Figs. 4 and 5 show the number of LSDB accessing in link and node protection with different real topologies. For link protection, both SymFTS and AsymFTS introduce much less computation overhead than existing proposals. For example, in AS1239 which has the largest number of nodes, AsymFTS reduces the number of LSDB accessing by 89.29%, 92.34%, 99.13% and 99.97%, compared to LFA, Tunnels, Uturns, and Notvia, respectively. In other topologies, AsymFTS achieves a similar improvement. Note that SymFTS may perform better than AsymFTS in some networks, e.g., in AS 1239, and AsymFTS may perform better than SymFTS in some networks, e.g., in AS 1221, since they adopt different approaches to identifying endpoints (see Section 4). AsymFTS normally achieves better performance in the networks with higher link density because it is easy to find an effective endpoint during rSPT computatoins. For example, AsymFTS introduces less overhead than SymFTS in AS1239 where link density per node is about 3.05, but SymFTS performs better in AS1221 where link density

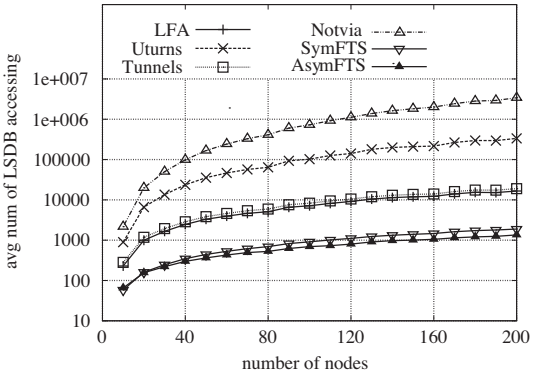


Fig. 6. Computation overhead with link protection in generated topologies.

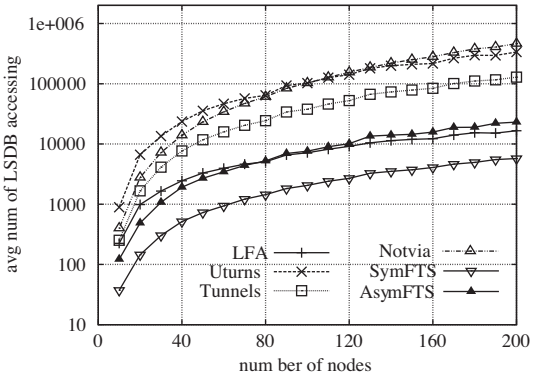


Fig. 7. Computation overhead with node protection in generated topologies.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

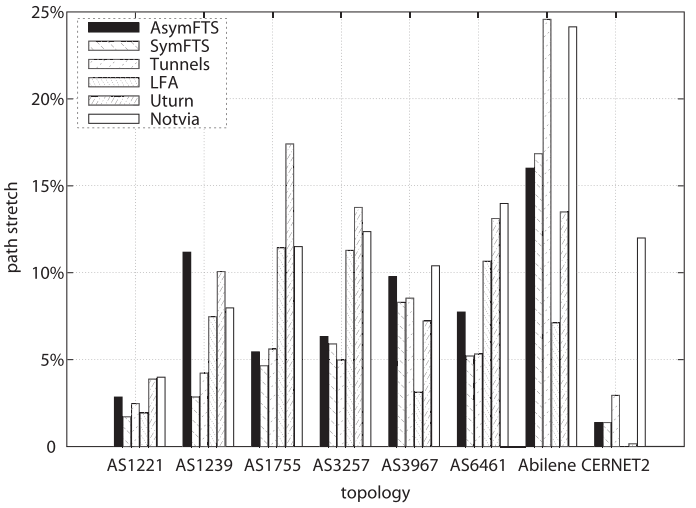


Fig. 8. Path stretch with link protection in real topologies.

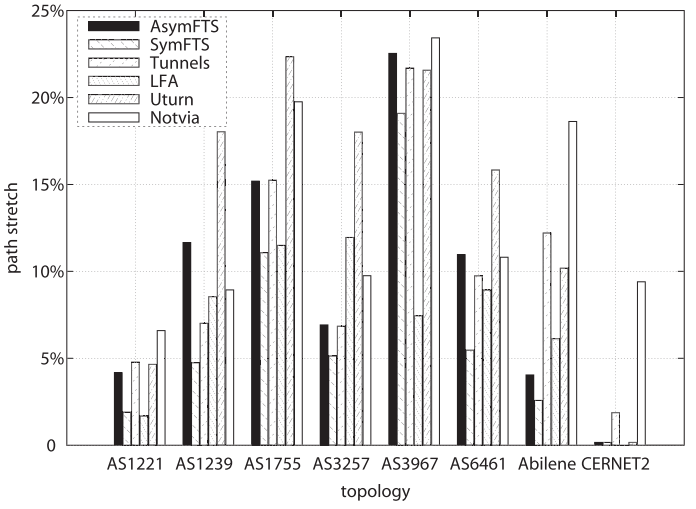


Fig. 9. Path stretch with node protection in real topologies.

per node is only 1.45. For node protection, expect that LFA achieves similar overhead compared to AsymFTS and SymFTS, the results are similar to link protection.

Figs. 6 and 7 show the number of LSDB accessing in link and node protection with different generated topologies. Compared to LFA, Tunnels, Uturns, and Notvia, AsymFTS reduces the number of LSDB accessing by 89.21%, 90.75%, 98.89%, 99.70%, respectively. For node protection, SymFTS introduces the least overhead and AsymFTS introduce a similar overhead to LFA. Thus, the overhead introduced by SymFTS and AsymFTS with node and link protection is much less than that by existing proposals. We believe the introduced overhead is acceptable for real deployment.

5.2. Path stretch

In this experiment, we evaluate forwarding path stretches introduced by forwarding packets with protection paths. We compare the length of a protection path with that of the new shortest path after a failure.

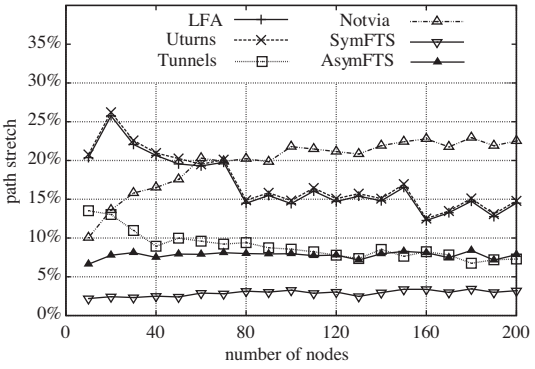


Fig. 10. Path stretch with link protection in generated topologies.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

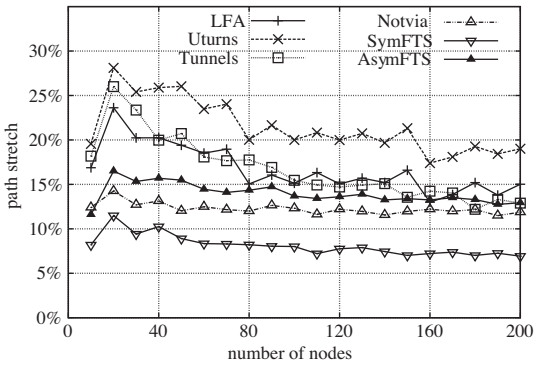


Fig. 11. Path stretch with node protection in generated topologies.

Figs. 8 and 9 show the path stretch results with link and node protection, respectively. Notvia needs to deliver packets to the other end of a failure and then forward them to the destinations. Therefore, the length of forwarding path of Notvia is longer. Similarly, random selections of alternate nodes in LFA and Uturns also increase the length of protection paths. The results demonstrate that the protection paths used by SymFTS and AsymFTS are short, because packets will not traverse a link or node more than once. SymFTS can use all the identified shortest alternate paths, and thus we observe that SymFTS has a shorter path stretch than AsymFTS in most cases.

Figs. 10 and 11 show the path stretch results in generated topologies. For link protection, the average path stretches introduced by LFA, Uturns, Tunnels, Notvia, SymFTS, and AsymFTS are 16.84%, 17.21%, 8.94%, 19.76%, 2.89%, and 7.79%, respectively. Similar results can be observed in node protection. LFA, Uturns, Tunnels, Notvia, SymFTS, AsymFTS introduce 16.69%, 21.44%, 16.70%, 12.26%, 8.11%, and 13.96% of the path stretch.

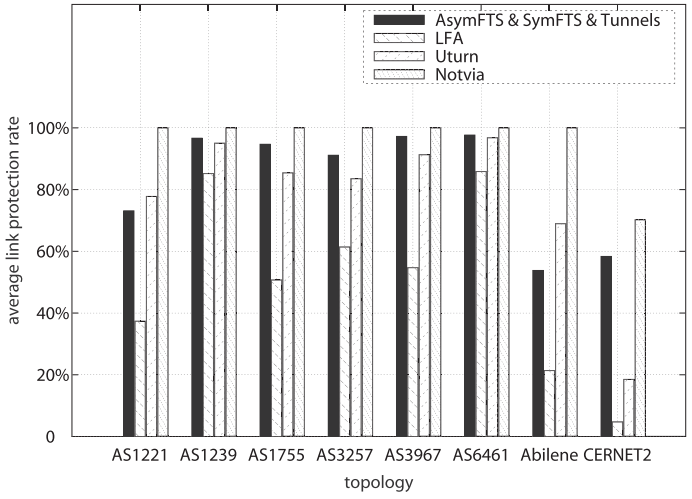


Fig. 12. Protection rate with link protection in real topologies.

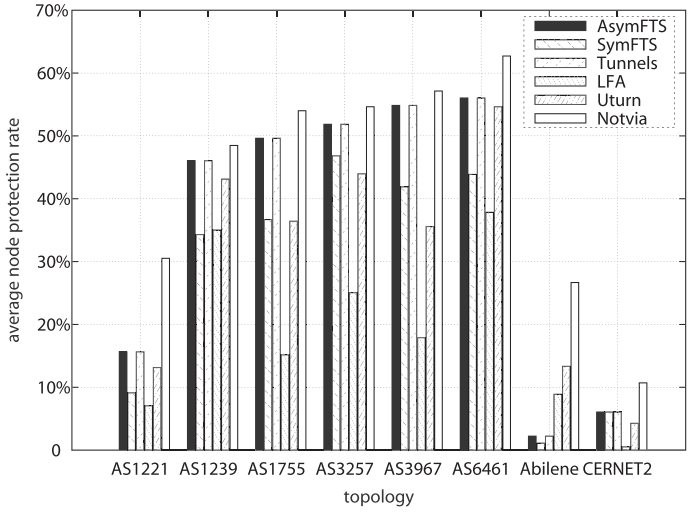


Fig. 13. Protection rate with node protection in real topologies.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

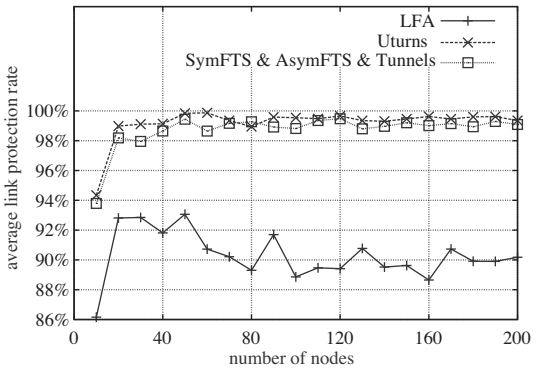


Fig. 14. Protection rate with link protection in generated topologies.

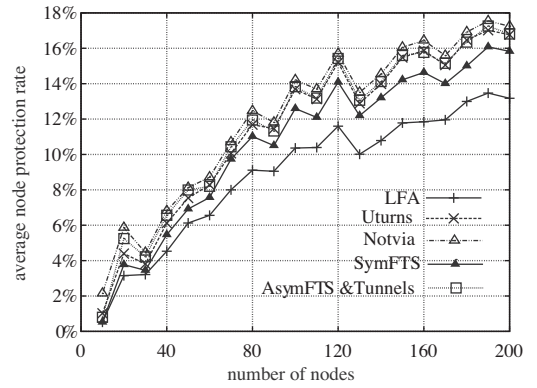


Fig. 15. Protection rate with node protection in generated topologies.

5.3. Protection rate

We evaluate protection ability of the approaches. Here, we use protection rate instead of failure coverage [26] as the metric because failure coverage only measures how many links or nodes can be protected without considering the usage of these components (i.e., links or nodes). We define the protection rate of the kth link/node, v_k, and average link/node protection rate λ, as follows:

$$\lambda = \sum_k \frac{\omega_k}{\sum_i \omega_i} \cdot v_k = \frac{\sum_k \delta_k}{\sum_i \omega_i}, \text{where } v_k = \frac{\delta_k}{\omega_k} \tag{12}$$

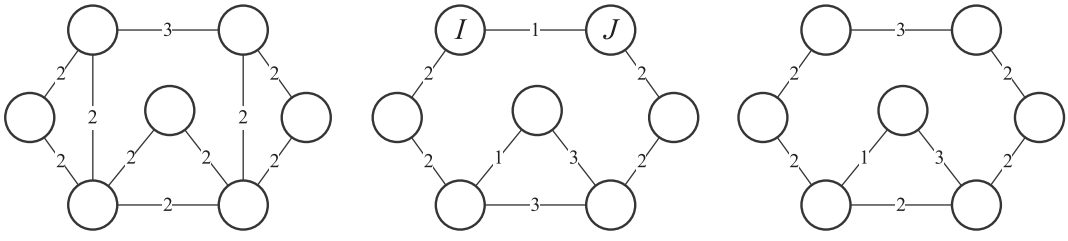
where ω_k denotes the number of shortest paths passing through the kth link/node, and δ_k denotes the number of paths which are not disrupted by the failure of the kth link/node using protection paths. Note that we consider protection for two direction traffic when we calculate δ_k. A protection path is successful only when the end-to-end node can communicate in two directions. Since the protection rate considers bidirectional connectivity for end-to-end communications, it is more accurate to evaluate the protection ability than the metric of failure coverage.

Figs. 12 and 13 show the average link and node protection rates of the approaches in real topologies. For link protection, SymFTS and AsymFTS achieve average 83% of link protection rates in the 8 topologies, respectively. The protection rates are similar to that of Notvia. For node protection, the average node protection rates are lower than link protection because node failures may incur isolation of the networks. However, AsymFTS has better performance than existing schemes, or at least has similar one to them..

Figs. 14 and 15 show the average link and node protection rates of the approaches in generated topologies. Since Notvia always achieves 100% link protection rate in the 2-connected networks, for simplicity, we do not put the result in Fig. 14. Link protection rates of LFA and Uturns are 90.28% and 99.18%, respectively, and Tunnels, SymFTS and AsymFTS achieve the same protection rate, 98.71%. The results demonstrate that SymFTS and AsymFTS achieve high protection ability. For node protection, LFA, Uturns, Tunnels, Notvia, SymFTS and AsymFTS achieve 8.93%, 11.50%, 11.66%, 12.13%, 10.64%, and 11.66% of node protection rates, respectively.

6. Discussion

Although different IPFRR approaches achieve different protection effectiveness with different required overhead, they can have their own application scenarios. A network operator can choose and deploy one of the approaches to trade off between overheads and performance according to network properties. In general, many network properties, e.g., network topologies and link weights, may impact the IPFRR performance. For simplicity, here we only use small scale network topologies to illustrate the impacts on protection effectiveness and discuss the application scenarios of these approaches. Note that the Uturns approach introduces much computation complexity but achieves similar performance to the Tunnels approach, and our proposed FTS improves the Tunnels approach. Thus, we do not discuss the Tunnels and Uturns approaches.



(a) A network that LFA can protect all single links

(b) A network that Notvia can protect all single links

(c) A network that FTS can protect all single links

Fig. 16. Example networks where the numbers beside each line denote link weights.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006

Table 2
Application scenarios of different approaches.

Approach	Network scale	Link weight difference	Link density
LFA	Any	Small	High
Notvia	Small	Any	Any
FTS	Any	Small	Any

The LFA approach needs rich network connectivity in order to provide full protection (which means a 100% protection rate). Densely connected networks with small link weight differences can be well protected by LFA [27]. In particular, a network with uniform link weights can be fully protected if each link is contained in at least one triangle which includes 3 links. It is not a necessary condition of full protection. However, such triangles may be still required in a network to ensure full protection with LFA. For example, the network in Fig. 16(a) can be fully protected by LFA. However, if we remove any link whose link weight is 2, the protection effectiveness of LFA will be weakened. Normally, LFA can protect large-scale networks, as long as the link densities are high.

The Notvia approach can provide full protection for networks with any link density and any link weight assignment. As shown in Fig. 16(b), Notvia can protect all single links, but neither LFA nor FTS can achieve it. For example, FTS cannot protect link $I \to J$. However, the configuration and computation overhead required by Notvia will increase significantly with the increase in network scale. Thus, Notvia may be suitable for some small-scale networks where LFA or FTS cannot achieve full protection.

The FTS approach can also perform well with any link density like Notvia. However, the overhead introduced by FTS is much less than Notvia, and thus FTS may be more suitable for large-scale networks. If we can properly adjust link weight assignments, FTS can achieve full protection with the same network. Fig. 16(c) shows an example of link weight assignments. In this example, FTS can protect all single links. We can observe that slight link weight adjustment leads to 100% protection. If a link weight assignment is well designed, e.g., any neighbor link has small link weight difference, FTS can achieve full protection. Table 2 summarizes the different application scenarios of LFA, Notvia and FTS.

7. Conclusion

In this paper, we propose a lightweight Fast Tunnel Selection (FTS) approach to implement tunnel-based IPFRR. FTS finds effective tunnel endpoints before the complete computation of entire SPTs and thus effectively reduces computation overhead. In particular, we present two FTS algorithms to protect networks with symmetric and asymmetric link weights, respectively. We simulate FTS with topologies of different sizes. The results show that FTS approach effectively reduces the computation overhead while high protection rates are still guaranteed.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC) Grant No. 61073166, 61133015 and 61161140454, the National Basic Research Program of China (973 Program) Grant No. 2009CB320502 and 2012CB315803.

Appendix A. Proof of Lemma 3.2

Proof. Prove by contradiction. Assume that $\exists R''$, s.t. $R'' \neq R' \wedge R'' \in \mathrm{SP}(I,R') \wedge R'' \in \mathrm{SP}(R',J)$, from which we obtain

$$\begin{cases} \mathrm{cost}(I,R'') + \mathrm{cost}(R'',R') = \mathrm{cost}(I,R'), \\ \mathrm{cost}(R',R'') + \mathrm{cost}(R'',J) = \mathrm{cost}(R',J). \end{cases}$$

Because $R' \in \mathrm{candidates}(J)$, we obtain

$$\begin{cases} \mathrm{cost}(I,R') < \mathrm{cost}(I,J) + \mathrm{cost}(J,R'), \\ \mathrm{cost}(R',J) < \mathrm{cost}(R',I) + \mathrm{cost}(I,J). \end{cases}$$

Such that

$$\begin{cases} \mathrm{cost}(I,R'') < \mathrm{cost}(I,J) + \mathrm{cost}(R'',J), \\ \mathrm{cost}(R'',J) < \mathrm{cost}(R'',I) + \mathrm{cost}(I,J) \end{cases}$$

$$\Rightarrow R'' \in \mathrm{cadidates}(J) \Rightarrow R'' \notin \mathrm{subtree}(I,J).$$

We can assume that $\mathrm{SP}(R'',J)$ consists of m nodes $R_0(=R'')$, $R_1, R_2, \ldots, R_m(=J)$. Obviously $R_m = J \in \mathrm{subtree}(I,J)$, so there is an integer i s.t. $0 \leqslant i < m \wedge R_i \notin \mathrm{subtree}(I,J) \wedge R_{i+1} \in \mathrm{subtree}(I,J)$. Because $R_i \in \mathrm{SP}(R'',J)$,

$$\begin{aligned} \mathrm{cost}(R_i,I) + \mathrm{cost}(I,J) &\geqslant \mathrm{cost}(R_0,I) + \mathrm{cost}(I,J) > \mathrm{cost}(R_0,J) \\ &\geqslant \mathrm{cost}(R_i,J) \end{aligned}$$

So that

$$\begin{aligned} &\mathrm{cost}(R_i,I) + \mathrm{cost}(I,J) > \mathrm{cost}(R_i,J) = \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(R_{i+1},J) \\ &= \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(R_{i+1},I) - \mathrm{cost}(I,J) \\ &\Rightarrow 2 \cdot \mathrm{cost}(I,J) > \mathrm{cost}(R_{i+1},I) - \mathrm{cost}(R_i,I) + \mathrm{cost}(R_i,R_{i+1}) \end{aligned}$$

This indicates that SymFTS will visit node R_i, and Inequation (6) is satisfied. However, SymFTS does not choose node R_i as the tunnel endpoint at last, so there is a node R s.t.

$$\begin{aligned} &\mathrm{cost}(I,R_i) + \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(I,R_{i+1}) - \mathrm{cost}(I,J) \geqslant \\ &\mathrm{cost}(I,R') + \mathrm{cost}(R',R) + \mathrm{cost}(I,R) - \mathrm{cost}(I,J) \\ &\Rightarrow \\ &\mathrm{cost}(I,R_i) + \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(I,R_{i+1}) \geqslant \\ &\mathrm{cost}(I,R') + \mathrm{cost}(R',R) + \mathrm{cost}(I,R). \end{aligned}$$

On one hand,

$$\begin{aligned} &\mathrm{cost}(I,R_i) + \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(I,R_{i+1}) \\ &= \mathrm{cost}(I,R'') + \mathrm{cost}(R'',R_i) + \mathrm{cost}(R_i,R_{i+1}) + \mathrm{cost}(I,J) \\ &\quad + \mathrm{cost}(J,R_{i+1}) \\ &= \mathrm{cost}(I,R'') + \mathrm{cost}(R'',J) + \mathrm{cost}(I,J). \end{aligned}$$

On the other hand,

$$\begin{aligned} &\mathrm{cost}(I,R') + \mathrm{cost}(R',R) + \mathrm{cost}(I,R) \\ &= \mathrm{cost}(I,R') + \mathrm{cost}(R',R) + \mathrm{cost}(J,R) + \mathrm{cost}(I,J) \\ &\geqslant \mathrm{cost}(I,R') + \mathrm{cost}(J,R') + \mathrm{cost}(I,J) \\ &= \mathrm{cost}(I,R'') + \mathrm{cost}(J,R'') + 2 \cdot \mathrm{cost}(R',R'') + \mathrm{cost}(I,J) \\ &\Rightarrow \\ &\mathrm{cost}(I,R'') + \mathrm{cost}(R'',J) + \mathrm{cost}(I,J) \\ &\geqslant \mathrm{cost}(I,R'') + \mathrm{cost}(J,R'') + 2 \cdot \mathrm{cost}(R',R'') + \mathrm{cost}(I,J) \\ &\Rightarrow \mathrm{cost}(R',R'') \leqslant 0 \end{aligned}$$

This is obviously false, so the initial assumption is false. This ends the proof. □

Appendix B. Proof of Lemma 4.1

Proof. Lemma 4.1 can be proved by contradiction. If R_j is not in candidate(J), then R_j is in either Pspace or Qspace, or neither of them. But R_i is in Qspace and R_j is downstream of R_i in rSPT(J), and thus R_j is in Qspace and not in Pspace. Therefore, the nodes in the shortest path between nodes J and R_j are not in Pspace, which means that R_i is not in Pspace. It contradicts to the fact that R_i is in Pspace. Therefore the initial assumption is false and R_j is in candidates(J). □

Appendix C. Proof of Lemma 4.2

Proof. Lemma 4.2 can be proved by contradiction. Assume that

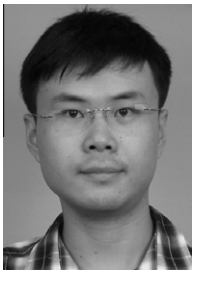
$$\exists R'', \text{ s.t.} R'' \neq R' \wedge R'' \in \mathrm{SP}(I, R') \wedge R'' \in \mathrm{SP}(R', J)$$

Therefore $\mathrm{cost}(R'', J) < \mathrm{cost}(R', J)$, which contradicts to the fact that node R' is the nearest to node J. □

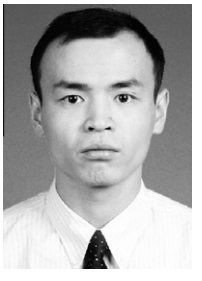
References

[1] Y. Yang, M. Xu, Q. Li, A lightweight IP Fast ReRoute algorithm with tunneling, in: Proc. of IEEE ICC, 2010.
[2] P. Francois, C. Filsfils, J. Evans, O. Bonaventure, Achieving subsecond IGP convergence in large IP networks, ACM SIGCOMM Computer Communication Review 35 (2005) 33–44.
[3] M. Shand, S. Bryant, IP Fast Reroute Framework (2009).
[4] D. Katz, D. Ward, Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop), June 2010.
[5] A. Li, P. Francois, X. Yang, On improving the efficiency and manageability of NotVia, in: Proc. of ACM CoNEXT, 2007.
[6] P.S.G. Enyedi, G. Retvari, A. Csaszar, IP Fast ReRoute: lightweight not-via without additional addresses, in: Proc. of IEEE Infocom, 2009.
[7] China Education and Research Network 2. <http://www.cernet2.edu.cn>.
[8] A. Atlas, A. Zinin, Basic Specification for IP Fast Reroute: Loop-Free Alternates, September 2008.
[9] A. Atlas, U-turn Alternate for IP/LDP Fast-Reroute, February 2006.
[10] S. Bryant, C. Filsfils, S. Previdi, M. Shand, IP Fast Reroute Using Tunnels, November 2007.
[11] S. Bryant, M. Shand, S. Previdi, IP Fast Reroute Using Notvia Addresses, July 2009.
[12] S. Rai, B. Mukherjee, O. Deshpande, IP resilience within an autonomous system: current approaches, challenges, and future directions, IEEE Communications Magazine 43 (2005) 142–149.
[13] K. Ho, N. Wang, G. Pavlou, C. Botsiaris, Optimizing post-failure network performance for IP Fast ReRoute using tunnels, in: Proc. of QShine, 2008.
[14] M. Menth, M. Hartmann, R. Martin, T. Cicic, A. Kvalbein, Loop-free alternates and not-via addresses: a proper combination for IP Fast Reroute?, Computer Networks 54 (2010) 1300–1315
[15] D. Hock, M. Hartmann, M. Menth, C. Schwartz, Optimizing unique shortest paths for resilient routing and Fast Reroute in IP-based networks, in: Proc. of NOMS, 2010.
[16] S. Kini, S. Ramasubramanian, A. Kvalbein, A. Hansen, Fast recovery from dual-link or single-node failures in IP networks using tunneling, IEEE/ACM Transactions on Networking 18 (6) (2010) 1988–1999.
[17] S. Nelakuditi, S. Lee, Y. Yu, Z. Zhang, C. Chuah, Fast local rerouting for handling transient link failures, IEEE/ACM Transactions on Networking 15 (2007) 359–372.
[18] S. Ray, R. Guerin, K.-W. Kwong, R. Sofia, Always acyclic distributed path computation, IEEE/ACM Transactions on Networking 18 (2010) 307–319.
[19] P. Francois, M. Shand, O. Bonaventure, Disruption free topology reconfiguration in OSPF networks, in: Proc. of IEEE INFOCOM, 2007.
[20] T.C.S.G.A. Kvalbein, A.F. Hansen, O. Lysne, Multiple routing configurations for fast IP network recovery, IEEE/ACM Transactions on Networking 17 (2009) 473–486.
[21] K.-W. Kwong, L. Gao, R. Guerin, Z. Zhang, On the feasibility and efficacy of protection routing in IP networks, in: Proc. of IEEE INFOCOM, IEEE Press, Piscataway, NJ, USA, 2010, pp. 1235–1243.
[22] Abilene. <http://abilene.internet2.edu/>.
[23] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring ISP topologies with rocketfuel, IEEE/ACM Transactions on Networking 12 (2004) 2–16.
[24] Brite. <http://www.cs.bu.edu/brite/>.
[25] O. Heckmann, M. Piringer, J. Schmitt, R. Steinmetz, Generating realistic ISP-level network topologies, IEEE Communications Letters 7 (2003) 335–336.
[26] M. Gjoka, V. Ram, X. Yang, Evaluation of IP Fast Reroute proposals, in: Proc. of IEEE COMSWARE, 2007.
[27] G.E.G. Retvari, J. Tapolcai, A. Csaszar, IP Fast ReRoute: loop free alternates revisited, in: Proc. of IEEE INFOCOM, 2011, pp. 2948–2956.

Mingwei Xu received the B.Sc. degree and the Ph.D. degree from Tsinghua University. He is a professor in Department of Computer Science at Tsinghua University. His research interest includes computer network architecture, high-speed router architecture and network security.



Yuan Yang received the Bachelor and the Master degree of Engineering in 2006 and 2009 respectively, from Tsinghua University, PR China. He is now a Ph.D. candidate at the Department of Computer Science & Technology in Tsinghua University. His major research interests include distributed routing protocol, computer network architecture and the next-generation Internet.



Qi Li received the B.Sc. degree from Tsinghua University, and the M.Sc. degree from Chinese Academy of Sciences, China. He is now a Ph.D. student in Department of Computer Science at Tsinghua University. He was a visiting Ph.D. student at the Chinese University of Hong Kong between 2009 and 2010. His research interest includes network architecture and protocol design, system and network security.

Please cite this article in press as: M. Xu et al., Selecting shorter alternate paths for tunnel-based IP Fast ReRoute in linear time, Comput. Netw. (2011), doi:10.1016/j.comnet.2011.11.006