This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

A Lightweight IP Fast Reroute Algorithm with Tunneling

Yuan Yang, Mingwei Xu, Qi Li
Department of Computer Science & Technology, Tsinghua University
{yyang, xmw, liqi}@csnet1.cs.tsinghua.edu.cn

***Abstract*—IP Fast ReRoute (IPFRR) has received increasing attention as a means to effectively shorten traffic disruption under failures. A major approach for IPFRR is to pre-calculate backup paths for nodes and links. Such approach is, however, hard to deploy due to the tremendous computational overhead. Thus, a lightweight IPFRR scheme is desired to effectively detour failures and provide routing protection. In this paper, we propose a Fast Tunnel Selection (FTS) algorithm to achieve tunnel-based IPFRR. The FTS algorithm can find an effective tunnel endpoint before complete computation of entire SPT and effectively reduce computation overhead. We simulate FTS with different size of generated topologies, and the results show that FTS algorithm reduces much computation overhead compared to existing approaches, and achieves a 99.10% average link protection rate and a 91.97% average node protection rate.**

I. Introduction

Fast evolution of network applications requires high availability and stability of Internet routing. However, routing as of today is not fully resilient to failures. A link or node failure will trigger routing convergence in 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 [1]. To address this issue, IP resilience to ameliorate fault recovery is gaining increasing attention.

IP Fast ReRoute (IPFRR) [10] is a potential technique to improve IP resilience in intra-domain routing, which can switch traffic to backup routes quickly and shorten the interruption period to tens of microseconds with the help of fast failure detecting techniques, such as bidirectional forwarding detection (BFD) [9]. 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 raise some problems, e.g., resource competition and router performance degradation.

To reduce the computation overhead, several improved tunnel-based IPFRR solutions are proposed. Li et al. reduce the number of shortest path tree (SPT) computations with the Notvia approach in IPFRR [4]. A router only needs to calculate a few SPTs. Enyedi et al. improve the Notvia approach by applying redundant trees [15]. Their proposal decreases the computing time by an order of magnitude, into the range of some few hundred milliseconds. However, the overhead of these proposals is still large because they still require recomputation of entire SPT for backup paths.

In this paper, we propose a lightweight backup path selection algorithm, called Fast Tunnel Selection (FTS), to achieve tunnel-based IPFRR. The FTS 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 once computation of the shortest path from the marked node to the node at the other side of the component finishes and the path does not include the component. Thus, FTS can find an effective tunnel endpoint before complete computation of the entire SPT and effectively reduce computation overhead. This actually provides a new point of view to compute the backup paths in IPFRR. Our simulation results show that the FTS algorithm greatly reduces the computation overhead compared to existing approaches, such as LFA [11], Uturns [12], Tunnels [13] and Notvia [14], and achieves much better protection rate than most existing approaches, e.g., FTS reduces at least 81% computation overhead compared to these approaches with link protection, and achieves a 99.10% average link protection rate.

The remainder of the paper is organized as follows. In Section II, we present background and related work. The FTS algorithm is proposed in Section III. We evaluate our algorithm in Section IV. Section V concludes the paper.

II. Background and Related Work

Reactive and proactive approaches are two major types of techniques for IP resilience [2]. Reactive approaches aim to shorten the convergence time of routing protocols, and proactive approaches, which are also called protection approaches, aim to reduce packet-loss by rerouting packets to backup paths.

Reactive approaches can deal well with multi-link failures. A typical approach is to shorten the convergence time for Interior Gateway Protocols (IGP) [1]. 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 only reroute 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 and guarantee traffic forwarding.

Several approaches are proposed to implement IPFRR, such as Loop Free Alternates (LFA) [11], Uturns [12], Tunnels [13], [16], and NotVia addresses [14]. Combination of LFA and Notvia is also proposed [17]. These approaches use tunnels

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

or directed forwarding to deliver packets in backup paths. The LFA approach utilizes neighbors' shortest paths to build protection paths. Thus, SPTs rooted at the neighbors must be computed. The Uturns approach considers neighbors' neighbors as alternate nodes to forward traffic, and 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. These tunnel endpoints are considered as virtual neighbors. The Tunnels approach also needs several 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 the failure. Thus, an SPT is to be computed for each notvia address. These approaches can cover most (or all) link and node failures alone. However, they all require computing several entire SPTs [18].

Existing improved tunnel-based IPFRR solutions reduce the computation overhead to some extent, but several SPT computations are still needed. Li et al. [4] indicate that protection paths do not traverse many nodes, and these nodes don't need to compute SPTs for these paths. However, a node still needs to compute for the paths which traverse it, especially when a failure is connected to the node. Enyedi et al. [15] decrease the number of notvia addresses by reformulating them in terms of redundant trees but re-computation of entire SPT for backup paths is still required. In our opinion, entire SPT computation may not be necessary for tunnel-based IPFRR.

III. Fast Tunnel Selection (FTS)

In this section, we will briefly review path selection in the traditional Tunnels approach. Then, we propose a lightweight IPFRR algorithm named FTS after analyzing the properties of tunnel endpoints. The notations used in this paper are presented in Table I.

TABLE I
Summary On Notations

Notation	Meaning
$Graph$	the network topology
I	the node that connected to a failure
J	a target node
N	a tunnel endpoint
$R1, R2, \ldots, Rn$	nodes in the network
$SP(Ri, Rj)$	shortest path from Ri to Rj
$SPT(Ri)$	shortest path tree rooted at Ri
$rSPT(Ri)$	reverse shortest path tree rooted at Ri
$cost(Ri, Rj)$	cost of shortest path from Ri to Rj
$P\text{-}space$	the set of nodes that can be reached from node I using normal forwarding without traversing link I-J
$Q\text{-}space$	the set of nodes that can reach node J using normal forwarding without traversing link I-J
$candidates(J)$	the set of candidate tunnel endpoints for target J

A. Finding protection paths in the Tunnels approach

In the Tunnels approach, node I is supposed to protect the links connected to it and its neighbor nodes. In order to find protection paths, target 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. After identifying targets, a tunnel endpoint must be identified for each target. A tunnel endpoint N is a node to which node I sends encapsulated packets when node I detects a failure between nodes I and J, and node N decapsulates the packets and enables normal 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 take protection of link I-J for example, Node N would be an effective tunnel endpoint when the following condition holds: $J \notin SP(I, N)$ & $I \notin SP(N, J)$. A tunnel endpoint selection algorithm proposed in the traditional Tunnels mechanism [13] identifies endpoints as follows:

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

Step 2. A set of nodes which can reach node J without traversing link I-J is calculated. This set is termed as $Q\text{-}space$ of node J with respect to link I-J. $Q\text{-}space$ 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-J.

Step 3. The set of candidate tunnel endpoints $candidates(J)$ is the intersection between $P\text{-}space$ and $Q\text{-}space$. The tunnel endpoint is selected from the set.

If the set is empty, Tunnels with directed forwarding can be used to protect this link. However, a special protocol is required to notify the tunnel endpoint of the hop for directed forwarding. Since the Tunnels approach can cover almost all failures, we do not discuss directed forwarding in this paper.

To protect a node, node J instead of link I-J is considered when calculating $P\text{-}space$ and $Q\text{-}space$. The procedure is similar to the above one, and we do not repeat it. As we can see from above, SPT computation consumes a lot of CPU cycles. There are two types of SPT here, i.e. $SPT(I)$ and $rSPT(J)$. As $SPT(I)$ is normally calculated and held by IGP such as OSPF and IS-IS, the main overhead is introduced by computing $rSPT(J)$. In fact, it is not necessary to calculate the entire rSPT and find $candidates(J)$ because what we require should be only one tunnel endpoint for each target.

B. Fast Tunnel Endpoint Selection

The main idea of FTS is to combine the tunnel endpoint selection process with the rSPT computation, in order that rSPT computation can be terminated as soon as we identify a tunnel endpoint. Note that the traditional Tunnels approach does not describe how to select a tunnel endpoint from $candidates(J)$. We discussed the key issue to quickly select one tunnel endpoint if there exists several candidates. For simplicity, we prescribe that the tunnel endpoint nearest to the target node J will be chosen, and we will analyze the properties of this node in Section III-C. This tunnel endpoint can be found by comparing costs of the candidate nodes.

However, we do not need to calculate $candidates(J)$ in FTS, because the nearest tunnel endpoint to node J can be identified before we finish calculating $rSPT(J)$. The procedure of FTS algorithm is as follows:

Step 1. P-*space* of node I is calculated with respect to link I-J, by marking all the nodes that are reached via link I-J with blue. Node I is also marked blue.

Step 2. Dijkstra algorithm is used to calculate reverse shortest paths from node J. Once the reverse shortest path to a node Ri is identified, the following steps should be taken.

Step 2.1. Node Ri is marked with red if the previous node of Ri is node I or it is marked red.

Step 2.2. Node Ri is selected as the tunnel endpoint for target J if Ri is neither marked blue nor marked red. Otherwise we should calculate shortest path for another node.

To choose tunnel endpoints, we propose FTS algorithm to check whether a node Ri meets the condition to be a tunnel endpoint as soon as node Ri is finalized, i.e. the shortest path from Ri to J is calculated. If Ri meets the condition, it is selected as the tunnel endpoint and the procedure halts. Otherwise, we should start a new round of shortest path calculation and tunnel endpoint check. FTS can find the tunnel endpoint nearest to node J, because the smallest-cost node which has not been calculated will be finalized during each round of calculation in Dijkstra algorithm. FTS algorithm is presented below. Inputs of the algorithm are $SPT(I)$, J and $Graph$. One node as the tunnel endpoint will be returned if there exists any, otherwise null is returned.

Algorithm 1 Fast Tunnel Selection Algorithm (FTS)

Require $SPT(I)$, J, $Graph$
mark all the nodes reached from node I via link I-J with blue, making use of $SPT(I)$
mark node I with blue
for all node Ri in $Graph$ **do**
  $distance[Ri] \Leftarrow$ infinity
  $previous[Ri] \Leftarrow$ undefined
$distance[J] \Leftarrow 0$
$Q \Leftarrow$ the set of all nodes in $Graph$
while Q is not empty **do**
  $u \Leftarrow$ node in Q with the smallest $distance[]$
  remove u from Q
  if $previous[u] = I$ **or** $previous[u]$ is red **then**
    mark u with red
  if u is not blue and not red **then**
    return u
  for all link Rj-u ingoing to u **do**
    if $distance[Rj] > distance[u] + cost(Rj, u)$ **then**
      $distance[Rj] \Leftarrow distance[u] + cost(Rj, u)$
      $previous[Rj] \Leftarrow u$
return null

C. Properties of Selected Tunnel Endpoint

The tunnel endpoint that FTS selected is the nearest one from target node J. We will analyze the properties of the selected tunnel endpoint.

Firstly, the computation overhead of FTS is related to topology, but we do not need to compute the entire $rSPT(J)$ in most cases. We can describe their distribution by the following theorem if there exist several tunnel endpoints [3].

Theorem 1: Node Rj is in $candidates(J)$ if node Ri is in $candidates(J)$, where Rj is downstream of Ri in $SPT(J)$.

Proof: The theorem can be proved by contradiction. If Rj is not in $candidate(J)$, then Rj is in either P-$space$ or Q-$space$, or neither of them. But Ri is in Q-$space$ and Rj is downstream of Ri in $SPT(J)$, and thus Rj is in Q-$space$ and not in P-$space$. Therefore, the nodes in shortest path between nodes J and Rj are not in P-$space$, which means that Ri is not in P-$space$. It contradicts to the fact that Ri is in P-$space$. Therefore the initial assumption is false and Rj is in $candidates(J)$. ■

Theorem 1 does not tell us whether a tunnel endpoint is near to node J actually. However, the number of candidate nodes that hide in the sub-tree rooted at a tunnel endpoint will increase quickly with the increase of the network size, which means that a lot of CPU cycles can be saved using FTS.

Secondly, protected packets will not be transmitted over two directions of any link using a selected tunnel endpoint. Let us take an example in Fig.1 where packets are transmitted over both directions of a link. Link costs are all set to 1 and $R2$ can be used as a tunnel endpoint to protect link I-J. Packets will traverse link $R1$-$R2$ and link $R2$-$R1$, and traverse node $R1$ twice, which causes extra bandwidth consumption and delay.

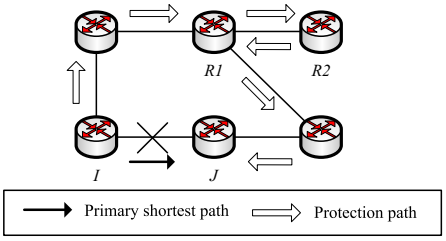

Fig. 1. Protecting link I-J using $R2$ as a tunnel endpoint

However, this problem will not occur with the FTS algorithm. If it is not true, there must be a node that receives each packet for two times, like node $R1$ in Fig.1. It is obvious that this node can be used as a tunnel endpoint. However, this node is nearer to node J than the original tunnel endpoint N, which contradicts to the fact that node N is the nearest endpoint to node J. As a result, FTS will never cause packets traversing a link or node more than one time. Although protection paths calculated by FTS may not be the shortest ones, packets will not retrogress anyway.

D. Improvement on Selecting Target Nodes

In the traditional Tunnels approach, one- and two-hop neighbors of node I are used as targets. However, this does not cover all protection cases in some topologies. For example, no tunnel endpoint can be found to protect link I-J or node J in Fig.2. However, we can use node $R1$ to deliver packets to node $R2$ which cannot be reached by normal forwarding when link I-J (or node J) fails. For the purpose of protecting such packet delivery, we improve the method of selecting target nodes in FTS as folllows.

One- and two-hop neighbors of node I are still used as targets. The child nodes of node Ri in $SPT(I)$ are used as targets if no tunnel endpoint can be found for target Ri. This kind of packet delivery is a special case, because most

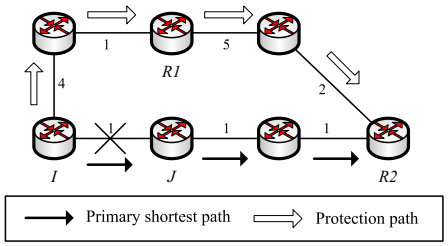

Fig. 2. Delivering packets to $R2$ using $R1$ when link I-J fails

links and nodes can be protected using the traditional Tunnels approach. However, the case arises for sure as we can see in our simulation in Section IV, and we suggest considering delivering packets as much as possible.

IV. Performance Evaluation

In this section, we evaluate FTS by simulation, compared with existing IPFRR approaches, such as LFA, Uturns, Tunnels and Notvia. We select alternate nodes randomly when there are several of them in LFA and Uturns, and use the nearest tunnel endpoint to node I in Tunnels. We consider end-to-end communications under every link/node failure using link/node protection in our simulation. We use BRITE [6] to generate different topologies for our simulation. The parameters required by BRITE are based on the description provided in [7], 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 will give the simulation results with both link protection and node protection.

Firstly we evaluate computation overhead of different approaches. Generally, the complexity of the shortest path first algorithm (Dijkstra) is $O(|E|log(|V|))$, and the basic operation during execution of Dijkstra algorithm is Link State Database (LSDB) accessing. Every LSDB accessing is a process of searching and accessing Link State Advertisements (LSA) announced by a router. LSDB accessing consumes the majority of CPU cycles and requires much more CPU cycles than other operations such as cost comparison and update. Thus, we evaluate the computation overhead by measuring the number of LSDB accessing.

Fig.3 shows the average number of LSDB accessing using LFA, Uturns, Tunnels, Notvia and FTS to protect links, respectively. FTS reduces 81.02%, 83.67%, 98.37%, 99.63% of overhead compared to LFA, Tunnels, Uturns, and Notvia, respectively. We can see in Fig.3 that the number of LSDB accessing of FTS has sawlike wave, which is mainly caused by the computation when no tunnel endpoint can be found for a target (see section III.D). Nevertheless, the overhead is small. Similarly, FTS introduces a few overhead with node protection. Fig.4 shows the result with node protection. The cost introduced by FTS is similar to LFA, and it reduces 92.03%, 77.04% and 90.40% of overhead compared to Uturns, Tunnels, and Notvia, respectively. Thus, the overhead of FTS with node and link protection is much less than most approaches except LFA with node protection and acceptable

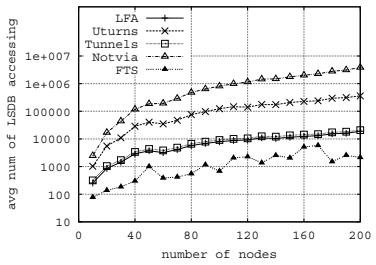

Fig. 3. Computation overhead comparison with link protection

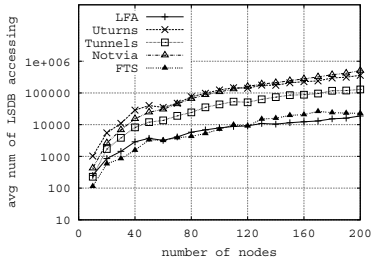

Fig. 4. Computation overhead comparison with node protection

for real deployment.

Secondly, we evaluate protection ability of the approaches. We use protection rate instead of failure coverage, as failure coverage only measures how many links or nodes can be protected, without considering the usage of these components. We define protection rate of the kth link/node, ν_k, and average link/node protection rate λ, as follows:

$$\lambda = \sum_k \nu_k \frac{\omega_k}{\sum_i \omega_i} = \frac{\sum_k \sigma_k}{\sum_i \omega_i}, \qquad where \qquad \nu_k = \frac{\sigma_k}{\omega_k},$$

where ω_k denotes the number of shortest paths, which pass through the kth link/node, and σ_k denotes the number of paths which are not disrupt by the failure of the kth link/node using protection paths. Note that bidirectional protection should be considered when we calculate σ_k. Protection is successful only when end-to-end nodes can communicate in two directions. Since protection rate considers paths traversing links/nodes as well as bidirectional connectivity for end-to-end communication, it is more accurate to evaluate protection ability than failure coverage.

Fig.5 shows the average link protection rates of LFA, Uturns, Tunnels and FTS in generated topologies. Since Notvia always achieves 100% link protection rate in 2-connected networks, we do not present the result in Fig.5. Link protection rates of FTS, Uturns, Tunnels and LFA achieves 99.10%, 99.08%, 99.07% and 92.09%, respectively. The simulation result shows that FTS achieves high protection ability which is close to full protection. FTS achieves a little higher link protection rate than Tunnels in some topologies. For instance, in the topologies with 130, 160 and 190 nodes, the link protection rate increases by 0.73%, 0.013% and 0.006%, respectively. These improvements are achieved by addressing the protection issue discussed in section III.D in FTS. Fig.6 shows the average node protection rates. We only consider the node failure cases that do not cause isolation of the network, and the result is similar to that with link protection. FTS, Uturns, Tunnels and LFA achieve 91.97%, 85.43%, 91.74%

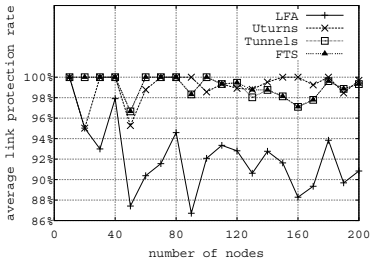

Fig. 5. Protection rate comparison with link protection

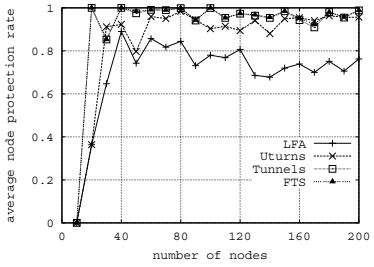

Fig. 6. Protection rate comparison with node protection

and 69.95% node protection rates, respectively, and FTS has the best protection rate among these approaches.

Thirdly, we evaluate forwarding path inflation. We compare the length of a protection path with that of new shortest path after a failure. Fig.7 shows the result when using link protection. The average path inflation ratio is 17.21% for LFA, 17.74% for Uturns, 8.39% for Tunnels, 19.02% for Notvia, and 7.59% for FTS. The result shows that FTS's protection paths are short, because FTS never makes packets traversing a link or node more than one time. Notvia must deliver packets to the other end of a failure first, and then to the destinations, so the length is long. Random selection of alternate nodes in LFA and Uturns also causes long length of protection paths. Fig.8 shows the result with node protection. The average path inflation ratio is 17.65% for LFA, 22.51% for Uturns, 16.91% for Tunnels, 12.59% for Notvia, and 14.19% for FTS.

V. Conclusion

Backup path calculation is an important technique in IPFRR. In this paper, we propose an FTS algorithm for tunnel-based IPFRR which introduces only a few computation overhead to protect links and nodes. A tunnel endpoint is selected once a node finishes the reverse shortest path which is from the target node to the endpoint. Therefore, we do not need to compute an entire SPT in most cases. Moreover, tunnel endpoints chosen by FTS effectively improve protection effectiveness. Simulation results show that FTS consumes much less computation overhead than existing approaches, such as LFA, Tunnels, Uturns, and Notvia, and achieves a 99.10% average link protection rate and a 91.97% average node protection rate. As future work, we will consider providing protection for multi-link failures with FTS, such as failures within Shared Risk Link Group [4].

Acknowledgment

This research is supported by National 863 project of China (No.2007AA01Z2A2, No.2009AA01Z251), National 973 project of China (No.2009CB320502), and Key Projects of National Science & Technology Pillar Program (No.2008BAH37B03).

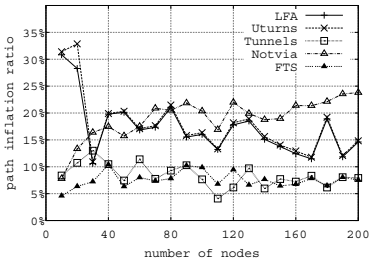

Fig. 7. Path inflation ratio with link protection

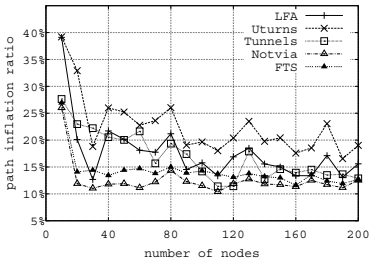

Fig. 8. Path inflation ratio with node protection

References

[1] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks", ACM SIGCOMM Computer Communication Review, 35(3):33–44, July 2005.
[2] S. Rai, B. Mukherjee, and O. Deshpande, "IP resilience within an autonomous system: Current approaches, challenges, and future directions", IEEE Communications Magazine, 2005,43(10):142–149.
[3] M. Gjoka, V. Ram, and X. Yang, "Evaluation of IP Fast Reroute Proposals", Proc. Of COMSWARE, Bangalore, India, Jan 2007.
[4] A. Li, P. Francois, and X. Yang, "On improving the efficiency and manageability of NotVia", Proc. of ACM CoNEXT, Dec 2007.
[5] P. Francois, "Improving the Convergence of IP Routing Protocols", PhD thesis, Université catholique de Louvain, Oct 2007.
[6] BRITE, http://www.cs.bu.edu/brite/.
[7] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz, "Generating realistic ISP-level network topologies", in IEEE COMMUNICATIONS LETTERS, 2003,7(7):335–336.
[8] S. Nelakuditi, S. Lee, Y. Yu, Z. Zhang, and C. Chuah, "Fast local rerouting for handling transient link failures", in IEEE/ACM Transactions on Networking, 15(2):359–372, Apr 2007.
[9] D. Katz and D. Ward, "Bidirectional Forwarding Detection", Internet Draft, draft-ietf-bfd-base-09.txt, Feb 2009.
[10] M. Shand and S. Bryant, "IP Fast Reroute Framework", Internet Draft, draft-ietf-rtgwg-ipfrr-framework-11.txt, June 2009.
[11] A. Atlas and A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates", RFC5286, Sep 2008.
[12] A. Atlas, "U-turn Alternate for IP/LDP Fast-Reroute", Internet Draft, draft-atlas-ip-local-protect-uturn-03.txt, Feb 2006.
[13] S. Bryant, C. Filsfils, S. Previdi, and M. Shand, "IP Fast Reroute using tunnels", Internet Draft, draft-bryant-ipfrr-tunnels-03.txt, Nov 2007.
[14] S. Bryant, M. Shand, and S. Previdi, "IP Fast Reroute Using Notvia Addresses", Internet Draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-04.txt, July 2009.
[15] G. Enyedi, G. Rétvári, P. Szilágyi, and A. Császár, "IP Fast ReRoute: Lightweight Not-Via without Additional Addresses", IEEE Infocom, Apr 2009.
[16] K. Ho, N. Wang, G. Pavlou, and C. Botsiaris, "Optimizing post-failure network performance for IP Fast ReRoute using tunnels", Proceedings of QShine, Hong Kong, July 2008.
[17] M. Menth, M. Hartmann, R. Martin, T. Cicic and A. Kvalbein, "Loop-Free Alternates and Not-Via Addresses: A Proper Combination for IP Fast Reroute?", Computer Networks, to appear.
[18] D. Hock, M. Hartmann, M. Menth, and C. Schwartz, "Optimizing Unique Shortest Paths for Resilient Routing and Fast Reroute in IP-Based Networks", Proceedings of NOMS, 2010.