

An Efficient Link Protection Scheme for Link-state Routing Networks

Haijun Geng^{*‡}, Xingang Shi^{†‡}, Xia Yin^{*‡}, Zhiliang Wang^{†‡}, Han Zhang^{*‡}

^{*}Department of Computer Science & Technology, Tsinghua University

[†]Institute for Network Sciences and Cyberspace, Tsinghua University

[‡]Tsinghua National Laboratory for Information Science and Technology (TNList)

Abstract—To enhance the network reliability without incurring significant extra overhead, we propose a novel link protection scheme, Hybrid Link Protection (HLP), to achieve failure resilient routing. Compared to previous schemes, HLP ensures high network availability in a more efficient way, and also provides other features such as load balancing. HLP is implemented in two stages. Stage one provides Multiple Next-hop Protection (MNP), where only one single Shortest Path Tree (SPT) needs to be constructed on each node to find multiple next hops for any destination. Stage two provides Backup Path Protection (BPP), where only a minimum number of links need to be protected, using special paths and packet headers, to meet the network availability requirement. We evaluate these algorithms in a wide spread of relevant topologies, both real and synthetic, and the results reveal that HLP can achieve high network availability without introducing conspicuous overhead.

Index Terms—backup path, availability, resilient routing

I. INTRODUCTION

With the rapid development of the Internet, more and more mission-critical and real-time applications, such as VoIP and video streaming, raise stringent network availability requirement. Unfortunately, network failures happen frequently and can also be caused by routine operations [1].

Two categories of solutions are proposed to deal with this problem. The first category uses reactive methods to accelerate the convergence of link-state routing protocol through tuning several parameters [2]. However, the risk of introducing instability in the network makes it less attractive, especially in the face of frequent momentary link failures. The other one adopts proactive schemes which compute backup paths in advance, so that packets can be forwarded over those precompute paths after the detection of a link failure. Existing proactive schemes can be further divided into two sub-categories, by whether special cooperation/signaling between routers are required for packet forwarding. Cooperation-free schemes compute multiple next-hops for each destination, and each router independently selects an appropriate next-hop for standard packet forwarding, where care must be taken such that the induced forwarding paths are loop-free. The benefit is that they can provide not only redundant backup links, but also other features such as load balancing and high throughput, while the problem is, they often can only protect a limited number of links. For example, recent studies [3] show that almost 40% links cannot be protected by LFA [4], a scheme adopted by the IP Fast-Reroute (IPFRR) [5] framework. The

other sub-category of schemes compute, for a link to protect, a multi-hop repair path that is agreed by all routers on that path. Thus special cooperation mechanisms have to be employed to reroute packets along that path. All these proactive schemes introduce considerable computation overhead, while only the latter sub-category can provide full coverage for a large network, at the cost of additional cooperation cost.

In this paper, we propose a novel scheme called Hybrid Link Protection (HLP), which meets the network availability requirement without inducing significant overhead by extending and combining the two kinds of proactive schemes above. HLP consist of two stages. Stage one provides Multiple Next-hop Protection (MNP), where a highly efficient SPT based algorithm is proposed to help a node find multiple loop-free next-hops for each destination, both locally and independently. Then, in stage two, based on the argument that uneven link failure probability or importance should be taken into account, Backup Path Protection (BPP) selects a minimum number of links according to the network availability requirement, and computes the corresponding multi-hop backup paths to protect them. In this way, HLP can attain the benefits of both kinds of schemes mentioned above, and achieves them with much less overhead.

The remainder of this paper is organized as follows. Section II introduces the related work. Section III presents the details of HLP's architecture, as well as the MNP and BPP algorithms. Section IV evaluates HLP in a variety of network topologies, and finally Section V concludes the paper.

II. RELATED WORK

Many schemes [4], [6], [7] that provide hop-by-hop loop-free routing are applicable in IPFRR (i.e., in the first category). Loop-free Alternate (LFA) [4] proposes several basic criteria for selecting a proper next-hop, including Loop-free Condition (LFC), Node Protection Condition (NPC) and Downstream Criterion (DSC), to guarantee loop-freeness. However, naively verifying these criteria requires multiple Shortest-Path Trees (SPT), and the cost increases proportionally to the degree of a node (one SPT for each neighbor). Routing deflection [6] relaxes DSC by taking nodes two hops away into account, at the cost of greater implementation complexity. TBFH [7] achieves faster computation by tightening DSC, but still needs to construct multiple SPTs.

The second category [8], [9] uses explicit cooperation/signaling schemes to route packets along a multi-hop repair path. Path splicing [8] creates a set of slices for the network based on random link-weight perturbations, and end system can control which slices the routers should use by embedding control bits in packet headers. Node/link-independent configurations of a topology are also computed to make the routing resilient under any single link fault [9]. The capability, as well as the complexity, of these algorithms is proportional to the number of alternative configurations they want to employ.

A few studies [3] [10] compare the computation complexity and network protection capability of the above two categories of schemes. Since both of them need intensive computation, it is argued that not-via address is better, in the sense that it provides higher network protection coverage [10]. In this paper, we focus on protection schemes that (1) are computationally efficient, (2) provide high network protection coverage, and (3) preserve as much as possible the benefits that multiple next-hop solutions have, such as no cooperation/signaling and load balancing.

III. HYBRID LINK PROTECTION

A. Network Model and Basic Architecture

We first explain the basic network model used in this paper. We represent the network by a undirected connected graph $G=(V, E)$, where V and E are respectively the set of nodes and links in the network. Each link $(u, v) \in E$ has a cost $L(u, v)$, and is further characterized by a failure probability $r(u, v)$. We also assume each node has a unique ID, $L(u, v) = \infty$ if u and v are not neighbors, and the link failure events are statistically independent of each other.

The main objective of HLP is to achieve high network availability by efficiently protecting links. We formally define the network availability $A(G)$ as follows (similar to that in [11]), and use it as a main metric to evaluate the protection capability of different schemes.

The end-to-end availability of a source-destination (s - d) pair is defined as the probability that the packets can be correctly forwarded from s to d . Assume that there exist n different forwarding paths from s to d , the i -th which is denoted by $p_i(s, d)$. Further, let the event that $p_i(s, d)$ works be denoted by $A_i(s, d)$, whose probability can be expressed as:

$$P(A_i) = \prod_{\forall (l, m) \in p_i(s, d)} r(l, m) \quad (1)$$

According to the Inclusion-Exclusion principle [12], the end-to-end availability of a source-destination pair can be expressed as:

$$A(s, d) = \sum_{k=1}^n (-1)^{(k-1)} S_k \quad (2)$$

where S_k denote the sum of the probabilities that a unique set of k paths from s to d are simultaneously working, which is further expressed as:

$$\begin{aligned} S_k &= \sum_{i < j < \dots < k} P(A_i \cap A_j \dots \cap A_k) \\ &= \sum_{i < j < \dots < k} \left(\prod_{(l, m) \in p_i(s, d) \cup p_j(s, d) \cup \dots \cup p_k(s, d)} r(l, m) \right) \end{aligned}$$

Then, the network availability can be computed as

$$A(G) = \frac{\sum_{s, d \in V, s \neq d} A(s, d)}{|V| * (|V| - 1)} \quad (3)$$

The main objective of our hybrid link protection (HLP) scheme is to explore path diversity and pre-compute backup paths, so that when link failures happen, a working path can be instantly activated to avert these links. HLP is implemented in two stages. Stage one (MNP) computes multiple next-hops for each destination, where a router only needs to construct a single SPT, and loop-freeness of the induced forwarding path can be guaranteed. Links remaining unprotected with MNP are handled in stage two, where BPP first identify such links and computes their individual contribution to the network availability (i.e., $A(G)$). Then it greedily selects a minimum number of key links to meet the network availability requirement, and computes the corresponding multi-hop backup paths.

In HLP, the node that detects a link failure is called the Failure Detecting Node (FDN). When a packet arrives at the FDN and its default next-hop is not available any longer due to the failure, FDN first checks whether there are multiple next-hops computed by MNP for the corresponding destination, and chooses any feasible one to forward the packet, if it exists. Otherwise, it uses a special header (e.g., using not-via address [13], source route, or any other appropriate signaling mechanism) to route the packet along the multi-hop repair path computed by BPP. Subsequent nodes receiving packets with the special header then directly forward it along the BPP path. At the same time, HLP initiates a control plane state synchronization, and re-execute MNP and BPP after a consensus on the network state is achieved by all nodes.

B. MNP

1) *Basic Idea*: Since each router carries out its computation independently, in the rest of the paper, the algorithm will be described with respect to a particular node c that performs such kind of computation.

The object of c is to compute a candidate set of next-hops $N_c(v)$ for each destination v ($v \neq c$), so that when packets destined to v arrive at c , c can select a next-hop from $N_c(v)$ to forward these packets to. In particular, we use $B_c(v)$ to represent the best/default candidate, which lies along the shortest path from c to v , and save it as the first entry in $N_c(v)$ (i.e., $B_c(v) = N_c(v)[0]$).

The node c builds a shortest path tree T_c rooted at itself, containing all the nodes in the network as potential destinations. We use $C_c(v)$ to denote the cost of the path from c to a specific node v in T_c , which is also the lowest cost from c

to v in the network. The parent of v in T_c is represented by $P_c(v)$. Since T_c is the shortest path tree, it is easy to verify that $B_c(v) = B_c(P_c(v)) = B_c(P_c(P_c(v))) = \dots$, that is, c will choose the same best next-hop for a node (i.e., a destination) and its descendents in T_c . To compute the other next-hops in $N_c(v)$, we start with a simple rule called Loop Free Condition (LFC) [4], which can be expressed as follows:

Theorem 1. For packets destined to a destination v , node c ($c \neq v$) can forward them to its neighboring node x if

$$C_x(v) < C_x(c) + C_c(v). \quad (4)$$

The resulting paths are loop-free and will reach the destination.

If we naively compute $C_x(v)$ on node c by constructing a SPT with x as its root, we will have to construct a SPT for each neighbor of c , and the cost will be particularly high if the degree of c is high. One of our contribution lies in the following rule, which is slightly more strict than the LFC rule:

Theorem 2. Given any two nodes u and v ($u, v \neq c$) in the shortest path tree T_c , let $D_c(v, u) = C_c(u) - C_c(B_c(u)) + L(u, v)$. If

$$D_c(v, u) < C_{B_c(u)}(c) + C_c(v), \quad (5)$$

we say u can contribute to v , and add $B_c(u)$ to $N_c(v)$. If packets destined to v are always forwarded by c to the next-hops in $N_c(v)$, the resulting paths are loop-free and will reach the destination.

Proof. $C_{B_c(u)}(v)$ is the cost from $B_c(u)$ to v in the SPT $T_{B_c(u)}$, and is also the lowest cost from $B_c(u)$ to v in the network. Since $C_c(u) - C_c(B_c(u)) + L(u, v)$ is the cost of a path from $B_c(u)$ to u to v , it must be no smaller than $C_{B_c(u)}(v)$, so $C_{B_c(u)}(v) \leq C_c(u) - C_c(B_c(u)) + L(u, v) < C_{B_c(u)}(c) + C_c(v)$. This satisfies equation (4) in Theorem 1, and thus the statement is true. $\square$

2) *Algorithm:* The detailed procedure of our Multiple Next-hop Protection (MNP) is provided in Algorithm 1, whose overall logic is very similar to the classic Dijkstra algorithm.

We assign a *visited* attribute to each node, so that only when a node is already in the tree, its *visited* attribute is *true*, and *false* otherwise. At the beginning, the costs of all nodes except c are set to infinity, and their *visited* attribute is set to *false*. Then the root node c is added to the SPT and set as the *current node* (lines 1 ~ 6). To expand the SPT, we go through several iterations. In each iteration, we examine the *unvisited* nodes and select one to add to the tree.

At the beginning of each iteration, an *unvisited* node v with the lowest tentative cost will be popped out from the priority queue Q by the EXTRACTMIN operation, where node ID is again used for tie breaking. It is then added to the tree, marked as *visited*, and used as the *current node* in the iteration, while its attributes like cost and parent are also set to a permanent value (lines 8 ~ 12). The corresponding best next-hop $B_c(v)$ is selected (lines 13 ~ 16), and more

candidates are added to the next-hop set $N_c(v)$ according to Equation (5) (lines 22 ~ 24). Note that when checking v against its neighbor u $N_c(u)$ can also be updated using the dual form of Equation (5) (lines 25 ~ 26).

Algorithm 1: MNP

```

1 for  $v \in V$  do
2    $C_c(v) \leftarrow \infty$ ;
3    $v.visited \leftarrow false$ ;
4  $c.visited \leftarrow true$ ;
5  $C_c(c) \leftarrow 0$ ;
6 ENQUEUE( $Q, < c, c, 0 >$ );
7 while  $Q$  is not empty do
8    $< v, p, tc > \leftarrow \text{EXTRACTMIN}(Q)$ ;
9   if  $v \neq c$  then
10     $v.visited \leftarrow true$ ;
11     $P_c(v) \leftarrow p$ ;
12     $C_c(v) \leftarrow tc$ ;
13    if  $P_c(v) = c$  then
14       $B_c(v) \leftarrow d$ ;
15    if  $P_c(v) \neq c$  then
16       $B_c(v) \leftarrow B_c(p)$ ;
17   for each neighbor  $u$  of  $v$  do
18     if  $u.visited = false$  then
19        $newdist \leftarrow C_c(v) + L(v, u)$ ;
20       if  $newdist \leq C_c(u)$  then
21         ENQUEUE( $Q, < u, v, newdist >$ );
22     if  $u.visited = true \wedge B_c(u) \neq B_c(v)$  then
23       if  $u$  contributes to  $v$  then
24         Add  $B_c(u)$  to  $N_c(v)$ ;
25       if  $v$  contributes to  $u$  then
26         Add  $B_c(v)$  to  $N_c(u)$ ;
```

3) *Complexity Analysis:* MNP maintains a priority queue Q , which stores the nodes together with their cost and parent. Let N denote the number of nodes which must change their cost or parent attributes (or both), M be the number of links that may cause any node in the queue to change its cost (which is performed by the decrease-key operation of the priority queue). Let en_Q be the time needed by ENQUEUE to enqueue a node, ex_Q be the time needed by EXTRACTMIN to extract the node, and dk_Q be the time needed by ENQUEUE (decrease-key) to update a node which is existing in the queue.

Theorem 3. The complexity of Algorithm 1 is $O(|V|lg|V| + |E|)$ when the queue is implemented as a Fibonacci heap.

Proof. Because each node must be enqueued into and extracted from the queue exactly once, and each link can cause at most one decrease-key operation, the total queue operation time in Algorithm 1 is at in $O(N * en_Q + N * ex_Q + M * dk_Q)$, where em_Q and dk_Q are times needed to perform extract minimum and decrease key operations in the Q , respectively. Beside queue operations, some manipulations are called at most twice for each of the M links, i.e., changing their next-hop sets, while other operations are called once for each of

the N nodes to set their attributes. Each of them can be implemented in constant time, and they cost $O(N + M)$ in total. So the time complexity of Algorithm 1 is still in $O(N * en_Q + N * ex_Q + M * dk_Q)$. Since there are at most S nodes in the queue, $en_Q = O(1)$, $ex_Q = \lg(N)$ and $dk_Q = O(1)$ when the queue is implemented as a Fibonacci heap, and the total time complexity is in $O(N * \lg(N) + M)$. By substituting N by $|V|$ and M by $|E|$, the complexity of Algorithm 1 is now $O(|V| * \lg(|V|) + |E|)$. $\square$

C. BPP

1) *Problem Formulation*: The multiple next-hops found by MNP can be utilized to detour around failed links. However, there is no guarantee that such backup next-hops can always be found, and thus the protection capability of MNP is limited (so do the other cooperation-free mechanisms). To further improve the network availability, we propose Backup Path Protection (BPP) to find backup paths when MNP fails to do so. Since forwarding along such paths needs additional cooperation/signaling mechanisms, we want to utilize them as little as possible.

Given a network $G(V, E)$ and the set of next-hops computed by MNP, our objective is to find a set of links and the corresponding backup paths, such that the network availability requirement ($A(G) \geq \mathcal{R}$, where $\mathcal{R}$ is a parameter chosen by network designers or operators) can be satisfied, while the number of such links is minimized.

2) *Algorithm*: Algorithm 2 illustrates the framework of Backup Path Protection (BPP). In the first step, each node in the network independently identifies its key links, which are links that cannot be protected by MNP.¹ A key link e is also associated with the forwarding paths that go through it, denoted by $e.P$ (lines 1 ~ 5). After MNP, these paths can be constructed using the next-hop sets on each node. We note that, here we only consider paths starting from the computing node instead of from any node in the network, and we will discuss how it affects the checking of the constraint $A(G) \geq \mathcal{R}$ later.

After that, the key links independently identified by each node are aggregated on a central node into a set K . The sets of associated forwarding paths of the same key link e are merged into a single $e.P$, and the contribution of protecting e to the network availability, $\delta(e)$, is computed based on $e.P$. Then the key links in K are sorted in a descendent order according to their contribution $\delta(e)$ (lines 6 ~ 11).

The procedure of computing $\delta(e)$ based on $e.P$ is as follows. Given a network $G(V, E)$, the contribution to the network availability of protecting a key link e is defined as:

$$\delta(e) = \sum_{\forall (s,d) \in e.P} A^*(s,d) - A(s,d) \quad (6)$$

where $A(s,d)$ is the end-to-end availability of a path (s,d) in G , as defined in Section III-A, while $A^*(s,d)$ is the end-to-end availability of a path (s,d) after e is protected by the backup path computed by BPP. The backup path BPP

¹A link $e = (v, B_v(d))$ is identified as a key link if $|N_v(d)| = 1$ (line 3).

Algorithm 2: BPP

```

// Each node  $v$  independently executes:
1  for  $d \in V$  do
2    if  $v \neq d \wedge |N_v(d)| = 1$  then
3       $e \leftarrow (v, B_v(d))$ ;
4       $e.P \leftarrow e.P \cup (v, d)$ ;
5      add  $e$  to  $\text{KeyLinks}(v)$  ;
// On a central node:
6  collect  $\text{KeyLinks}(v)$  from each  $v \in V$ ;
7   $K = \cup_{v \in V} \text{KeyLinks}(v)$ ;
8  for  $e \in K$  do
9    merge  $e.P$ ;
10   compute  $e$ 's contribution  $\delta(e)$  by Equation (6);
11  sort  $K$  in the descendent order of  $\delta(e)$ ;
12   $K' = \emptyset$ ;
13  compute  $A(G)$ ;
14  while  $A(G) < \mathcal{R}$  and  $K \neq \emptyset$  do
15    select the first link  $e$  from  $K$ ;
16     $A(G) \leftarrow A(G) + \delta(e) / (|V| * (|V| - 1))$ ;
17    add  $e$  to  $K'$ ;
18    remove  $e$  from  $K$ ;
19  distribute  $K'$  to each node in the network;
```

computes for a link e is simply the new shortest path in the graph $G \setminus e$. When computing $A^*(s,d)$, we augment G by adding a pseudo link (s,d) between s and d to get a new graph G^* . The availability $r(s,d)$ of this pseudo link is just the product of the availability of all links along the new shortest path, and $A^*(s,d)$ is computed in the same way as $A(s,d)$ using Equation (2), but on the augmented graph G^* instead of G .

In this way, we model the different contribution to the network availability of protecting different key links. Then we simply select the key links from K one by one, in the order of their contributions, until the network availability requirement $A(G) \geq \mathcal{R}$ is satisfied, or the set K is exhausted (lines 12 ~ 18). Finally, these key links are distributed to all nodes in the network, so that these nodes can reach a consensus on protecting which links using special cooperation/signaling mechanisms. The backup path can be easily computed by each node, and can also be distributed by the central node.

At last, we explain when computing $e.P$, the associated forwarding paths of e , why we only add those paths that start from the computing node v (line 4) and the impact it brings on satisfying the network availability constraint. The reason we do this is simply for reducing the computational complexity, since enumerating all paths in the network that go through e will be $|V|$ times more time consuming. On the other hand, since $\delta(e) = \sum_{\forall (s,d) \in e.P} A^*(s,d) - A(s,d)$ is composed of the individual contribution of each path $(s,d) \in e.P$, i.e., $A^*(s,d) - A(s,d)$, which cannot be negative, we can only underestimate the contribution of protecting e by conservatively adding paths to $e.P$. Thus, when BPP considers $A(G) \geq \mathcal{R}$ to be already satisfied, the constraint is indeed satisfied.

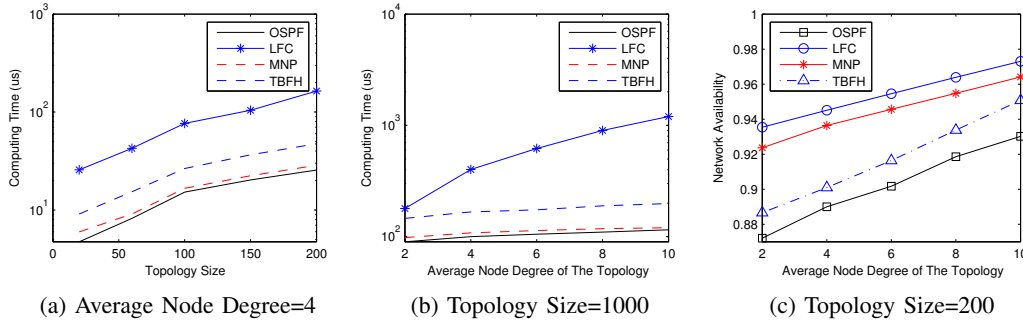


Fig. 1: Results on Different Topologies

IV. PERFORMANCE EVALUATION

The objective of our performance evaluation is twofold:

i) To indicate the effectiveness of MNP, including its computation complexity and its effect on the network availability. To this end, we compare MNP with the standard OSPF and some cooperation-free mechanisms mentioned in Section II, including LFC [4] and TBFH [7]. Other schemes mentioned there are much more complex, and none of them can provide near 100% network availability, so are not compared here.

ii) To demonstrate the improvements achieved by HLP, which combines MNP and BPP. For this purpose, we compare HLP with the strategy that uses BPP only, and another strategy that uses cooperation-free mechanisms to achieve a full protection. For the latter one, we use the incremental OSPF algorithm to compute a multi-hop repair path for each link. Other cooperation-free mechanisms mentioned in Section II are more complex, and cannot effectively improve the network availability from our requirement (i.e., $\geq 99.99\%$) anymore, so are not directly compared here.

To evaluate our algorithm, we use the real Abilene network (11 nodes, 14 links), and four other ISP topologies inferred by Rocketfuel [14], including Exodus (79 nodes, 147 links), Telstra (108 nodes, 153 links), Tiscali (161 nodes, 328 links) and Sprint (315 nodes, 972 links). We also generate synthetic topologies with BRITE [15], using parameters as listed in Table I. For similarity, we use a simple model to characterize link failure events. The fail probability of each link e is randomly generated in the range from 0 to 0.02. All the simulations are conducted on a PC with Intel i5 CPU at 1.7GHz and 1.5G Memory.

TABLE I: Parameters for BRITE

Model	N	HS	LS
Waxman	20-1000	1000	100
m	NodePlacement	GrowthTypem	α
2-10	Random	Incremental	0.35
β	BWDist	BwMin	BwMax
0.65	Heavy Tail	100.0	1024.0

A. MNP Performance

1) *Computation Time*: The computation time of MNP, OSPF, LFC and TBFH on different ISP topologies is listed in

Table II, where the number is the time averaged on all nodes in the corresponding network, and the lower, the better. It is obvious that MNP's performance is close to that of OSPF, and they both outperform the other two schemes, especially when the network is larger. For example, in the Sprint topology, OSPF uses 140.23 μ s, MNP uses 150.12 μ s, while TBFH and LFC's complexity are two and six times higher, respectively. This is due to the single SPT construction in MNP, whose complexity is in the same order of that in OSPF.

TABLE II: Computation time for Real Topologies

	Network	Computation time (μ s)			
		OSPF	LFC	TBFH	MNP
Real	Abilene	6.82	7.27	6.97	6.83
Measured	Exodus	44.36	128.29	88.76	50.23
	Telstra	49.45	163.43	100.34	54.34
	Tiscali	79.45	398.34	183.56	83.45
	Sprint	140.23	906.78	368.12	150.12

The impact of the network topology on the computing time is further illustrated in Fig. 1 using synthesized topologies generated by BRITE. Fig. 1(a) and 1(b) depict the corresponding computation time of each scheme, in a log-scale manner, when the topology size or the average node degree of the topology changes, respectively. When the topology size or the average node degree increases, the average computation time increases accordingly, indicating a similar trend as that in Table II. In a large network with dense connections, i.e., when $|V| = 1000$ with an average node degree of 10, LFC and TBFH may be several orders slower than MNP, whose performance keeps close to OSPF. Fig. 1(b) also shows a remarkable influence of the average node degree on LFC, which computes a SPT for each neighbor, while TBFH needs to compute a few SPTs.

TABLE III: Network Availability for Real Topologies

	Network	Network Availability(%)			
		OSPF	LFC	TBFH	MNP
Real	Abilene	93.27	97.12	94.45	96.89
Measured	Exodus	84.54	91.25	86.76	90.01
	Telstra	85.34	93.23	87.34	92.11
	Tiscali	82.12	92.65	84.45	90.89
	Sprint	81.76	93.25	83.45	90.13

2) *Network Availability*: Table III provides the network availability provided by each protection scheme, on the five real ISP topologies. From the results, we can see that MNP has an clear advantage over TBFH and OSPF, and only performs slightly worse than LFC. For example, in Sprint, MNP and

LFC can achieve a network availability of greater than 90%, while OSPF and TBFH can only provide an availability around 80%.

We also investigate how the protecting capabilities of the four schemes, when the topology density increases. As shown in Fig. 1(c), when the average node degree increases, all schemes provide better protection results, while MNP and LFC are always much better than OSPF and TBFH.

B. HLP

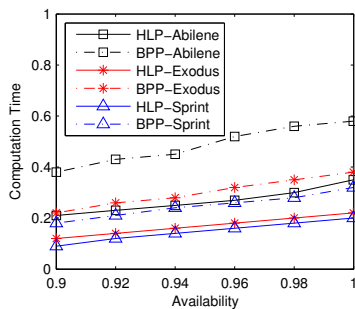


Fig. 2: Computation Time for Real Topologies

Due to space limitations, we only present the real topology results, while the results for synthetic topologies are similar.

Fig. 2 shows the computation time of HLP and BPP, normalized by the computing time achieved by a full protection with multi-hop backup paths, on different topologies. The largest network availability we test is 99.99%, which is close to the theoretical maximum value, 100%. HLP has a clear advantage over BPP, both of which perform much better than a naive full-protection. For example, HLP typically needs around 20% time of that of full protection, while BPP costs around 40% time of a full protection. The advantage of BPP over the naive full-protection verifies the effect of intelligently selecting a small number of links according to their contribution to the network availability, while the advantage of HLP over BPP verifies the effect of taking the advantage of the much less computational intensive MNP.

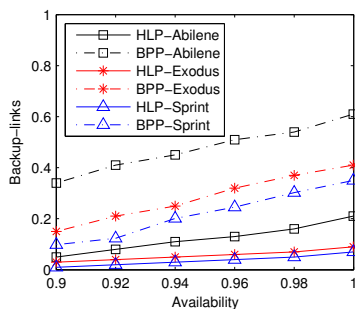


Fig. 3: Backup-links for Real Topologies

Fig. 3 shows more details. In the figure, we plot the number of key links that each scheme must use multi-hop backup paths to protect, under different levels of network availability

requirement. The y -axis (Backup-links) is the ratio of the number of protected key links to the number of all links in the network. When the topology size or the density increases, the performance of HLP improves more upon other schemes. The reason is that more multiple next-hops are found in MNP, which effectively reduces the number of links to be protected in BPP.

V. CONCLUSION AND FUTURE WORK

We presented HLP as a novel link protection scheme to achieve high network availability for link-state routing networks. We described how HLP improves network availability by combining MNP and BPP. In the future, we will focus our research on designing effective link failures models. We are convinced that our work will be more fruitful under more accurate prediction of link failure event.

VI. ACKNOWLEDGE

This work is supported by China 863 program (Grant No. 2015AA010201), National Natural Science Foundation of China (Grant No. 61402253 and 61202357), and Project for 2012 Next Generation Internet technology research and development, industrialization, and large scale commercial application of China (Grant No. 20121763).

REFERENCES

- [1] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 749 – 762, 2008.
- [2] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second igp convergence in large ip networks," *Computer Communication Review*, vol. 35, no. 3, pp. 35 – 44, 2005.
- [3] P. Francois and O. Bonaventure, "An evaluation of ip-based fast reroute techniques," in *CoNEXT 2005 - Proceedings of the 2005 ACM Conference on Emerging Network Experiment and Technology*, 2005, pp. 244 – 245.
- [4] E. A. Atlas and E. A. Zinin, "Basic specification for ip fast reroute: Loop-free alternates, rfc 5286," 2008.
- [5] S. B. M. Shan, "Ip fast reroute framework, rfc 5714," 2010.
- [6] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *SIGCOMM*, 2006, pp. 159–170.
- [7] P. Mérindol, P. Francois, O. Bonaventure, S. Cateloin, and J. J. Pansiot, "An efficient algorithm to enable path diversity in link state routing networks," *Comput. Netw.*, vol. 55, no. 5, pp. 1132–1149, Apr. 2011.
- [8] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," in *SIGCOMM*, 2008, pp. 27–38.
- [9] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent directed acyclic graphs for resilient multipath routing," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 153–162, 2012.
- [10] M. Menth, M. Hartmann, R. Martin, T. Čičić, and A. Kvalbein, "Loop-free alternates and not-via addresses: A proper combination for ip fast reroute?" *Computer Networks*, vol. 54, no. 8, pp. 1300 – 1315, 2010.
- [11] R. Terruggia, "Reliability analysis of probabilistic networks," *PhD Thesis*, 2010.
- [12] W. Feller, "An introduction to probability theory and its applications vol. i," Wiley, 1968.
- [13] M. S. S. Bryant, S. Previdi, "A framework for ip and mpls fast reroute using not-via addresses, rfc 6981," 2013.
- [14] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, pp. 2 – 16, 2004.
- [15] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings*, 2001, pp. 346–353.