A Hybrid Link Protection Scheme for Ensuring Network Service Availability in Link-state Routing Networks

Haijun Geng, Han Zhang, Xingang Shi, Zhiliang Wang, Xia Yin, Ju Zhang, Zhiguo Hu, and Yong Wu

Abstract: **The internet is playing an increasingly crucial role in both personal and business activities. In addition, with the emergence of real-time, delay sensitive and mission-critical applications, stringent network availability requirement is put forward for internet service providers (ISPs). However, commonly deployed intra-domain link-state routing protocols react to link failures by globally exchanging link state advertisements and recalculating routing table, inevitably causing significant forwarding discontinuity after a failure. Therefore, the loop-free criterion (LFC) approach has been widely deployed by many ISPs for coping with the single network component failure scenario in large internet backbones. The success of LFC lies in its inherent simplicity, but this comes at the expense of letting certain failure scenarios go unprotected. To achieve full failure coverage with LFC 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. HLP is implemented in two stages. Stage one provides an efficient LFC based method (MNP-e). The complexity of the algorithm is less than that of Dijkstra's algorithm and can provide the similar network availability with LFC. Stage two provides backup path protection (BPP) based on MNP-e, 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. HLP not only needs around 10% time of that of full protection, but also provides full protection capabilities that full protection provide.**

Index Terms: **Incremental shortest path first, loop-free, multipath routing, network availability, network link failure**

Manuscript received November 7, 2018; approved for publication by Mubashir Husain Rehmani, Division III Editor, November 8, 2019.

This work is partially supported by the National Key Research and Development Program of China (No.2018YFB1800401), the National Natural Science Foundation of China (No.61702315), Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications)(SKLNST-2018-1-19), the National Natural Science Foundation of China (No.61872226) and the Natural Science Foundation of Shanxi Province, China (No.201701D121052).

H. Geng is with the School of Software Engineering, Shanxi University, Open Foundation of State Key Laboratory of Networking and Switching Technology, email: ghj123025449@163.com.

H. Zhang is with the School of Cyber Space and Technology, Beihang University, email: zhhan@buaa.edu.cn.

X. Shi and Z. Wang are with the Institute for Network Sciences and Cyberspace, Tsinghua University, and Beijing National Research Center for Information Science and Technology, email: {shixg, wzl}@cernet.edu.cn.

X. Yin is with the Department of Computer Science and Technology, Tsinghua University, and Beijing National Research Center for Information Science and Technology, email: yxia@tsinghua.edu.cn.

J. Zhang is with the School of Software Engineering, Shanxi University, Open Foundation of State Key Laboratory of Networking and Switching Technology, email: zj4090@139.com.

Z. Hu is with the School of Computer and Information Technology, Shanxi University, email: huzhiguotj@sxu.edu.cn.

Y. Wu is with the School of Software Engineering, Shanxi University, email: wuyong@sxu.edu.cn.

Z. Wang is the corresponding author.

Digital Object Identifier: 10.1109/JCN.2019.000056

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 [1], [2]. Unfortunately, network availability is often reduced by unexpected failures as well as routine operations [3]. Traditional routing algorithms, such as open shortest path first (OSPF) and Intermediate system-to-intermediate system (IS-IS), are mostly concerned with finding shortest paths towards each destination, and thus cannot provide good connectivity under frequent network failures. On detecting a failure, they start a global link-state advertisement and then recompute routes, inevitably causing network outage [4], [5]. This highlights the need for mechanisms that possess fast and efficient recovery capability [6]–[9].

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 [10]. 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 must be 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 [11] show that almost 40% links cannot be protected by loop free alternates (LFA) [12], a scheme adopted by the IP fast-reroute (IPFRR) [13] 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 consists of two stages. Stage one provides MNP-e, where a highly efficient incremental shortest path first (i-SPF) based algorithm is proposed to help a node find all the LFC next-hops for each destination, both locally and independently. Based on the existing work on this research area, we for the first time propose an algorithm whose complexity is less than that of Dijkstra algorithm and without degrading the network availability of LFC. 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. In this way, HLP can attain the benefits of both kinds of schemes mentioned above, and achieves them with much less overhead.

Our main contributions are summarized as follows:

- We propose HLP to effectively protect failed links. HLP can provide full network protection in a very efficient way.
- We design MNP to compute multiple loop-free next-hops for link protection, where only a single SPT need to be constructed on each router. This, both in theory and practice, is much faster than existing algorithms that need to construct multiple SPTs.
- We design MNP-e (an extension of MNP) to compute multiple loop-free next-hops for link protection, theoretical analysis indicates that the computation complexity of MNP-e is less than that of constructing a shortest path tree, and MNP-e can provide the similar network availability with LFC. We theoretically prove the correctness of MNP-e algorithm.
- We argue that different link failures have different contribution to the network availability, and propose BPP to protect a minimal number of links to meet the network availability requirement.
- We evaluate the performance of our algorithm against other link protection schemes in a unified framework, using both real and synthetic topologies with different network availability requirements. Compared with other solutions, HLP can not only ensure good network availability, but also cannot introduce conspicuous overhead.
- These advantages make HLP a good candidate for both traditional telecommunication networks andemerging complex networks that require failure repair and load balancing in a highly dynamic environment.

The remainder of this paper is organized as follows. Section II introduces the related works. Section III presents the details of HLP's architecture. Section IV describes the MNP-e and its properties in detail. Section V introduces BPP. Section VI evaluates MNP-e and HLP in a variety of network topologies, and finally Section VII concludes the paper.

II. RELATED WORKS

Nowadays, network failures have become routine events rather than exceptions. Many schemes have been proposed to deal with this problem from different aspects, from physical level methods such as optical routing protection, to IP level approaches.

IETF has drafted a framework named IPFRR [12]–[14], which aims to provide fast recovery from network failures. The basic idea is that, when a node detects the failure of a link directly connected to itself, it can immediate switch to backup paths that are specifically computed for this failure. Based on this basic framework, current solutions fall into two categories. The first category computes multiple next-hops for each destination such that, even if routers independently select them in a hop-by-hop manner for detouring the failed links, loop-free can still be guaranteed, and thus packet routing/forwarding can be done as usual, i.e., no further cooperation or signaling mechanisms are needed. The second category compute a multi-hop repair path for any link to be protected, and requires explicit cooperation/signaling between routers to ensure packets are indeed routed along this path when necessary.

Firstly, we will describe some schemes that can provide hop-by-hop loop-free routing are applicable in IPFRR (i.e., in the first category). Equal-cost multipath routing (ECMP) [15] allows packets to be forwarded along multiple paths of equal cost, which can be specifically tuned by network operators. However, ECMP cannot offer good reliability since it is limited to cases where equal cost paths exist. Loop-free alternate (LFA) [12] proposes several basic criteria for selecting a proper next-hop, including loop-free criterion (LFC), node protection condition (NPC) and downstream criterion (DC), 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 [16] relaxes DC by taking nodes two hops away into account, at the cost of greater implementation complexity. TBFH [17] achieves faster computation by tightening DC, but still needs to construct multiple SPTs. Permutation Routing [18], [19] treats routers as a sequence of resources, and creates permutations of these resources that offer several forwarding alternatives, where each permutation is equivalent to a SPT and the time complexity is proportional to the number of permutations they want. Several algorithms [20]–[22] compute a directed acyclic graph (DAG) for each destination to avoid loops, so in a network with $|V|$ nodes, their time complexity is in the order of $|V|$ times the cost of computing a single DAG, which is already more complicated than computing a SPT. More next-hops can be found at the cost of increasingly sophisticated DAGs, but there is no guarantee to find an alternate next-hop for each destination (or each link). FIR [21] improves the protecting capability against a single link failure by computing different alternates for packets from different incoming ports, at the cost of an increased computation overhead, and a restricted scenario, i.e., only for protecting a link. In [23], authors show that improving LFA failure case coverage is feasible without touching the physical topology and the forwarding paths in any ways, or requiring any new features from the IP data and the control planes that are essentially fixed by what is available in commercial network gear today. FIFR++ approach [24] employ progressive link metric increments in conjunction with FIFR and proves that FIFR++ is loop-free while protecting against single link failures. Authors in [25] present DMPA-e by incorporating hop count, a topology dependent metric. However, DMPA-e cannot achieve full failure coverage against single network component failures. Then, we will describe some schemes that uses explicit cooperation/signaling schemes to route packets along a multi-hop repair path. Multi-topology routing [26], [27] compute multiple routes based on backup network topologies tailored for specific failures, either by removing the corresponding links or by increasing their associated weights. Routers can control which topology the routers should be employed by changing additional bits in the packet header. Path splicing [28] 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 [29].

The capability, as well as the complexity, of these algorithms is proportional to the number of alternative configurations they want to employ. Failure-carrying packets (FCP) [30] carries link failure information in the IP packet header to allow routers to diagnose problems and select alternate paths. However, this scheme also requires considerable overhead to find the new working path when receiving a packet carrying root-cause failure messages. Not-via [14] proposes a framework to use special not-via addresses to provide link protection by such multi-hop paths. Authors in [31] propose the scheme to cope with single network component failure in segment routing networks. Authors in [32] propose how to deploy LFA into software defined networks (LFASDN). And they suggest to build an augmented fat-tree topology which allows LFA to protect against all single link and node failures. TOD [4] proposes tunneling on demand (TOD) to handle one or dual link failures, but needs an additional signaling protocol to establish tunnels. Authors in [33] presented the STATIC-ROUTING-RESILIENCY problem and explored the power of static fast failover routing in a variety of models. And also the schemes can provide resiliency against k-1 failures, with limited path stretch. A few studies [11], [34] 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 [34]. 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. Therefore, we propose a novel link protection scheme, hybrid link protection (HLP), to achieve failure resilient routing. Our previously published paper [35] gives some preliminary ideas about HLP. HLP is implemented in two stages. Stage one performs a multipath routing algorithm, which is similar to the paper [16]. However, the time complexity of the stage one increases proportionally to the degree of a node, which makes it impractical to deploy on the internet. Therefore, we improved the stage one of HLP in our published paper [36]. In [36], stage one provides an efficient LFC based method (MNP). The time complexity of MNP does not depend on the degree of the calculating router. However, MNP cannot compute all the feasible backup next-hops which are conformed to LFC. This paper mainly concerns on how to overcome the drawbacks of [35] and [36]. In this paper, we for the first time propose an algorithm (MNP-e) in the stage one whose complexity is less than that of Dijkstra algorithm and without degrading the network availability of LFC. A comparison of several existing fast reroute solutions and ours are summarized in Table 1.

III. OVERVIEW OF HYBRID LINK PROTECTION

A. Network Model

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 $r(u, v)$. $L(u, v) = \infty$ if u and v are not neighbors, and the link failure events are statistically independent of each other. We use T_c to denote the shortest path tree rooted at node c, $D(T_c, x)$ to denote the descendants of node x (x is included) in the T_c. $T'_c(c, x)$ is a shortest path tree rooted at node c when the edge $(c, x) \in T_c$ change its weight to $-L(c, x)$. $D(T'_c(c, x), x)$ to denote the descendants of node x (x is included) in the $T'_c(c, x)$.

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, 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 ith which is denoted by $p_i(s, d)$. We also use $P_i(s, d)$ to represent the set of links on the $p_i(s, d)$. Further, let the event that $p_i(s, d)$ works be denoted by $A_i(s, d)$, whose probability can be expressed as [37]:

$$P(A_i) = \prod_{\forall (m,n) \in P_i(s,d)} r(m, n). \tag{1}$$

According to the Inclusion-Exclusion principle [38], the end-to-end availability of a source-destination pair can be expressed as:

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

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 [37]:

[1] D is average node degree of the network.
[2] m is the number of slice.

Table 1. Comparison of different routing algorithms.

Algorithm	Key enhancements	Drawbacks	Loop holes	Simulator used	Time complexity
ECMP [15]	(1) Allows packets to be forwarded along multiple paths of equal cost (2) Compatible with current routing protocol (3) Can offer load balancing	Cannot deal with all single link failure scenarios	Yes	OSPFD	$O(\|E\|\cdot \lg(\|V\|))$
LFA [12]	(1) Allows packets to be forwarded to alternative neighbors which are conformed to LFA (2) Compatible with current routing protocol (3) Can offer load balancing (4) Does not require any support from other routers	Cannot deal with all single link failure scenarios	Yes	Own computation framework which is conducted on a PC	$D{\cdot}O(\|E\|\cdot \lg(\|V\|))^1$
TBFH [17]	(1) Combine fast rerouting and load balancing (2) Compatible with current routing protocol	(1) Cannot deal with all single link failure scenarios (2) Cannot calculate all the next hops that satisfy LFA	Yes	Own computation framework which is conducted on a PC	$2{\cdot}O(\|E\| \cdot \lg(\|V\|))$
FCP [30]	(1) Eliminate the convergence process completely (2) Can deal with all single link failure scenarios	(1) Need to modify the current routing protocol (2) Need to modify the header of packet (3) Cannot offer load balancing	No	OSPFD	Authors do not analyze the time complexity of the algorithm
Path splicing [28]	(1) Combine multiple routing trees to each destination over a single network topology (2)Compatible with current routing protocol	(1) Cannot deal with all single link failure scenarios (2) Cannot offer load balancing	Yes	Own computation framework which is conducted on a PC	$m{\cdot}O(\|E\|\cdot \lg(\|V\|))^2$
TOD [4]	(1) Propose a model for interface specific routing (2) Can deal with all single link failure scenarios	(1) Need to modify the current routing protocol (2) Need to modify the header of packet (3) Cannot offer load balancing	No	Own computation framework which is conducted on a PC	$O(\|E\|\cdot\|E\|\cdot \lg(\|V\|))$
LFASDN [32]	(1) Provide loop detection for LFAs in OpenFlow-based networks (2) Can deal with all single link failure scenarios	(1) Only for SDN network (2) Cannot offer load balancing	No	Own computation framework which is conducted on a PC	Authors do not analyze the time complexity of the algorithm
MNP [36]	(1) Can calculate the majority of the next hops that satisfy LFA (2) Compatible with current routing protocol (3) Can offer load balancing	(1) Cannot deal with all single link failure scenarios (2) Cannot calculate all the next hops that satisfy LFA	Yes	Own computation framework which is conducted on a PC	$O(\|E\|\cdot \lg(\|V\|))$
MNP-e [25]	(1) Can calculate all the next hops that satisfy LFA (2) Compatible with current routing protocol (3) Can offer load balancing	Cannot deal with all single link failure scenarios	Yes	Own computation framework which is conducted on a PC	$O(\|E\|\cdot \lg(\|V\|))$
HLP	(1) Efficiently combine two categories of routing protection algorithms (2) Compatible with current routing protocol (3) Can offer load balancing (4) Can deal with all single link failure scenarios	Need to use special address (e.g., not-via address)	No	Own computation framework which is conducted on a PC	$O(\|E\|\cdot \lg(\|V\|))$

$$\begin{aligned} S_k &= \sum_{i<j<\cdots<k} P(A_i \cap A_j \cdots \cap A_k) \\ &= \sum_{i<j<\cdots<k} \left(\prod_{(m,n)\in P_i(s,d)\cup P_j(s,d)\cup\cdots\cup P_k(s,d)} r(m,n) \right). \end{aligned}$$

Then, the network availability can be computed as

$$A(G) = \frac{\sum\limits_{s,d\in V, s\neq d} A(s,d)}{|V|\cdot(|V|-1|)}. \quad (3)$$

B. Basic Architecture

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 computes multiple next-hops for each destination based on LFC, where loop-freeness of the induced forwarding path can be guaranteed. Links remaining unprotected with stage one 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 stage one 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 [14], source route, or any other appropriate signaling mechanism) to route the packet along the multihop 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 stage one and BPP after a consensus on the network state is achieved by all nodes.

IV. EFFICIENT LOOP-FREE CRITERION BASED SCHEMES

A. *Problem Description*

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. If the node c and v have direct links, then we have $C_c(v) = L(c, v)$. 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 $B_c(v) = B_c(P_c(v)) = B_c(P_c(P_c(v))) = \cdots$, that is, c will choose the same best next-hop for a node (i.e., a destination) and its descendants in T_c. To compute other next-hops in $N_c(v)$, we start with a simple rule called loop free criterion (LFC) [12] , 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

$$C_x(v) < C_x(c) + C_c(v). \tag{4}$$

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

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.

However, for the actual deployment on the internet, a LFC-based scheme should introduce a small additional burden on the current deployed routing protocol. This paper is dedicated to finding an efficient LFC-based scheme which is suitable for deploying on an ISP network. In particular, we focus on the following problem: *Given a computing node c and T_c, can we find an efficient LFC-based algorithmic technique and the algorithm conforms to the following two conditions: (1)The time complexity of the algorithm is less than constructing a shortest path tree. (2) It can provide the same network availability with LFC.* In this paper we propose two LFC-based methods, multiple next-hop protection (MNP) and an extension of multiple next-hop rotection (MNP-e). We will describe both of the algorithms and their properties in detail in the following sections.

B. *MNP*

The problem proposed above can be come down to how to cleverly compute $C_x(v)$ in the T_c. One of our contributions lies in the following rule, which is slightly more strict than the LFC rule:

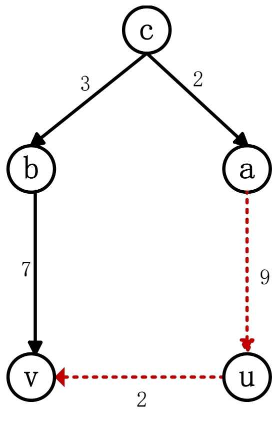


Fig. 1. An example for explaining the Theorem 2.

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

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

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 (4) in Theorem 1, and thus the statement is true. □

The condition in (5) is a little more strict than that in (4), however, checking whether it is satisfied by two nodes u and v can be accomplished in a much simpler way as follows. When constructing the SPT T_c, whenever we insert a new node u, we only need to verify (5) against a node v that is u's neighbor and is already in T_c, since if u and v are not neighbors, $L(u, v) = \infty$ and it is not possible to satisfy (5). In this way, we can compute $N_c(v)$ for any node v by constructing a single SPT, which is much faster than other multipath algorithms.

We will use an example to illustrate the Theorem 2. Fig. 1 is a shortest path tree rooted at c. In this figure, note that $D_c(v, u) = C_c(u) - C_c(B_c(u)) + L(u, v) = 9 + 2 = 11$, while $C_{B_c(u)}(c) + C_c(v) = 2 + 10 = 12$. Therefore, we can get $D_c(v, u) < C_{B_c(u)}(c) + C_c(v)$. From Theorem 2, we say u can contribute (its best next-hop a) to v. From Theorem 2, we can see that node c can use a as a validate next hop to v.

B.1 Algorithm

The detailed procedure of our 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.

To determine which node will be added to the tree, for each $unvisited$ node u, we first compute a tentative parent $p_c(u)$ and a tentative cost $tc_c(u)$ with respect to the tree as follows. Considering some $visited$ node p in the tree, the cost of the shortest path starting from c, going through p, to the $unvisited$ node u is $c_c(p) + L(p, u)$. Among all possible p's, the one achieves the smallest value of the cost $c_c(p) + L(p, u)$ is chosen as the tentative parent of u, i.e., $p_c(u)$, where node ID of p is used as a tie breaker. The corresponding tentative cost is just $tc_c(u) = c_c(p_c(u)) + L(p_c(u), u)$, which represents the lowest cost from c to u, using existing shortest paths in the tree. In our implementation, we save u together with its $p_c(u)$ and $tc_c(u)$ in a priority queue Q using the ENQUEUE operation: if u already exists in Q, it will be updated only when its new tentative cost is smaller, or if the cost is the same but its new tentative parent has a smaller ID. In each iteration, only $unvisited$ neighbors of the $current\ node$ need to update their tentative parent and cost in Q (lines 18–21), since $visited$ nodes are already in the tree, and have been assigned a permanent cost and parent.

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 (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 (5) (lines 25–26).

B.2 Example

We use a simple step-by-step example to illustrate how MNP works, as shown in Fig. 2, where node c dynamically constructs the SPT and computes the next-hops for all other destinations.

Since $P_c(b) = P_c(a) = P_c(d) = c$, we can get $B_c(b) = b$, $B_c(a) = a$, and $B_c(d) = d$ when b, a and d are added to the tree. When node e is added to the tree, e will choose b as its parent, and we get $B_c(e) = B_c(b) = b$. [1]

When e is added to the tree, $B_c(a) = a$ and $C_c(e) + C_a(c) = 9$, so $D_c(e, a) = C_c(a) - C_c(B_c(a)) + L(a, e) = 4$. Since $D_c(e, a) < C_c(e) + C_a(c)$, therefore a can contribute to e, so $B_c(a) = a$ is added to $N_c(e)$.

Because $B_c(e) = b$, $C_c(a) + C_b(c) = 7$, $D_c(a, e) = C_c(e) - C_c(B_c(e)) + L(e, a) = 7$. Since $D_c(a, e) = C_c(c) + C_b(c)$, so e cannot contribute to c.

Similarly, when e is added to the tree, $B_c(d) = d$ and $C_c(e) + C_d(c) = 10$, so $D_c(e, d) = C_c(d) - C_c(B_c(d)) + L(d, e) = 5$. Since $D_c(e, d) < C_c(e) + C_d(c)$, d can contribute to e, so $B_c(d) = d$ is added to $N_c(e)$.

Because $B_c(e) = b$, $C_c(d) + C_b(c) = 7$, $D_c(d, e) = C_c(e) - C_c(B_c(e)) + L(e, d) = 8$. Since $D_c(d, e) > C_c(d) + C_b(c)$ and e cannot contribute to d.

[1] The best next-hop $B_c(\cdot)$ is just the first element in the next-hop set $N_c(\cdot)$.

Algorithm 1 MNP.

```
Input:
    Network graph $G = (V, E)$ and $T_c$
Output:
    $N_c(v), (\forall v \in V \wedge v \neq c)$
1:  for $v \in V$ do
2:      $C_c(v) \leftarrow \infty$
3:      $v.visited \leftarrow false$
4:  end for
5:  $c.visited \leftarrow true$
6:  $C_c(c) \leftarrow 0$
7:  ENQUEUE($Q, < c, c, 0 >$)
8:  while $Q$ is not empty do
9:      $< v, p, tc >\leftarrow$EXTRACTMIN(Q)
10:     if $v \neq c$ then
11:         $v.visited \leftarrow true$
12:         $P_c(v) \leftarrow p$
13:         $C_c(v) \leftarrow tc$
14:         if $P_c(v) = c$ then
15:             $B_c(v) \leftarrow d$
16:         end if
17:         if $P_c(v) \neq c$ then
18:             $B_c(v) \leftarrow B_c(p)$;
19:         end if
20:     end if
21:     for each neighbor $u$ of $v$ do
22:         if $u.visited = false$ then
23:             $newdist \leftarrow C_c(v) + L(v, u)$
24:             if $newdist <= C_c(u)$ then
25:                 ENQUEUE($Q, < u, v, newdist >$)
26:             end if
27:         end if
28:         if $u.visited = true \wedge B_c(u) \neq B_c(v)$ then
29:             if $u$ contributes to $v$ then
30:                 Add $B_c(u)$ to $N_c(v)$
31:             end if
32:             if $v$ contributes to $u$ then
33:                 Add $B_c(v)$ to $N_c(u)$
34:             end if
35:         end if
36:     end for
37: end while
38: return $N_c(v), (\forall v \in V \wedge v \neq c)$
```

B.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 $O(|E| \cdot \lg(|V|))$

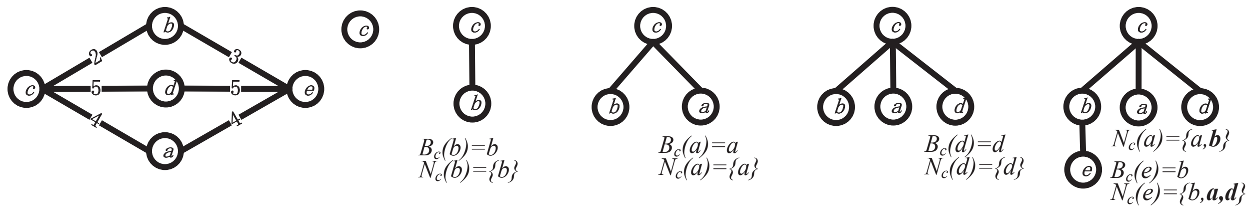


Fig. 2. Step by step construction of the SPT rooted at node c and the next-hop sets.

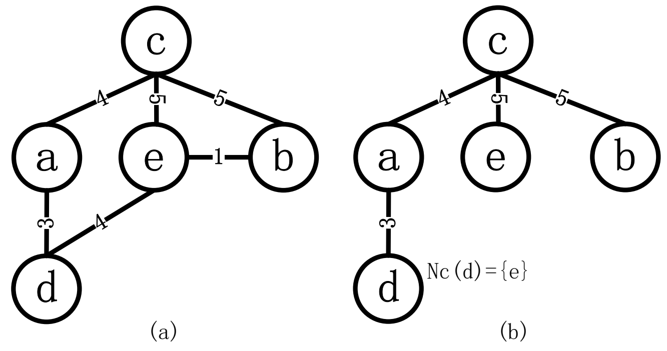


Fig. 3. An example for explaining the drawback of MNP.

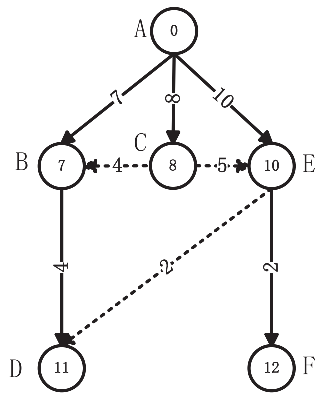


Fig. 4. Shortest path tree rooted at A before link change.

when the queue is implemented as a 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 $O(N{\cdot}en_Q + N{\cdot}ex_Q + M{\cdot}dk_Q)$. 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 $O(N{\cdot}en_Q + N{\cdot}ex_Q + M{\cdot}dk_Q)$. 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 heap, and the total time complexity is in $O(|M|lg|N|)$. By substituting N by $|V|$ and M by $|E|$, the complexity of Algorithm 1 is now $O(|E| \cdot \lg(|V|))$. □

C. An Extension of MNP (MNP-e)

Fig. 3 gives an example to explain the drawback of MNP. Fig. 3(a) is a simple network topology which consists of 5 node and 6 edges. Fig. 3(b) is a shortest path tree rooted at node c. From Fig. 3(b), we can get $C_b(d) = 5 < C_c(d) = 7$, therefore node b is a feasible backup next-hop from c to d. However, we cannot find this feasible backup next-hop employing MNP. This drawback is due to that node d only considers its neighbors' best next-hop as its potential backup next-hop. The time complexity of MNP does not depend on the degree of the calculating router. However, the network availability of MNP is lower than that of the LFC. Based on the existing work on this research area, we for the first time propose an algorithm whose complexity is less than that of Dijkstra's algorithm and without degrading the network availability of LFC.

We will first discuss incremental shortest path first (i-SPF) algorithm, which is the foundation of MNP-e algorithm. The OSPF and IS-IS routing protocols which are widely deployed in today's internet calculate a shortest path tree (SPT) from each router to other routers in an autonomous system (AS). Lots of commercial routers have adopted dynamic SPT algorithms which employ the structure of the previously computed SPT rather than recomputed an new SPT from scratch when the network topology changes. The reason is that when network topology changes, the new computed SPT does not differ conspicuously from the old one.

The i-SPF algorithm is carried out as following steps:
(1) Finding all the potentially affected nodes and marking them *floating*.
(2) Computing the potential new distance, parent and the difference between the old distance and the potential new distance $\triangle$ for the potentially affected nodes.
(3) In each iteration, a node with the smallest $\triangle$ (least positive or most negative) is selected, and then a subtree, instead of only one node, is appended to the new SPT. All of the above nodes are marked *anchored*.

Perhaps the most intuitive way to demonstrate iSPF is through an example. Consider the network topology depicted in Fig. 4, which is composed of 6 nodes and 8 links. The letter besides the node is its label. The solid lines are the links in the shortest path tree which is rooted at node A, while the dotted lines are the links not in the above shortest path tree. The number inside

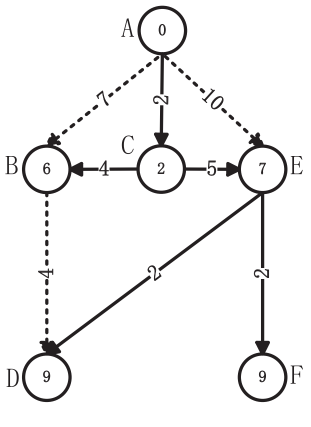


Fig. 5. Shortest path tree rooted at A when the weight of (A, C) is changed to 2.

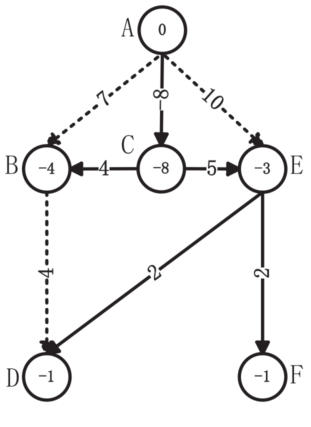


Fig. 6. Shortest path tree rooted at A when the weight of (A, C) is changed to -8

the circle is the shortest distance from node A to that node.

At some point, the weight of the edge (A, C) is changed from 8 to 2. Nodes from B to F are all affected nodes and are marked as *floating*. Only the root node A is marked as *anchored*. For all the *floating* nodes, we check whether they have links to the *anchored* nodes, and calculate the new distance and the change ($\triangle$) in distance, which gives queue $\{(B, 7, 0), (C, 2, 6), (E, 10, 0)\}$ where the third value is the $\triangle$. In the next iteration, $(C, 2, 6)$ will be considered first since it has the largest $\triangle$. C will be marked as *anchored*. Since C has an outing edge to E and B, the new distance and $\triangle$ of B and E is calculated, which gives queue $\{(B, 6, 1), (E, 7, 3)\}$. Then E is selected, instead of only marking E as *anchored*, the original subtree rooted at E is considered together. Therefore the node F is selected and marked as *anchored* at the same time. Since E has an outing edge to D, the new distance and $\triangle$ of D is calculated, which gives queue $\{(B, 6, 1), (D, 9, 2)\}$. Since node D has the largest potential shortest distance decrease, node D is marked as *anchored*. In the next iteration, node B is selected. The new shortest path tree is depicted in the Fig. 5.

Here we will describe a special example, when a single link changes its weight to the opposite number. For example, the weight of the edge (A, C) is changed from 8 to -8. Only the root node A is marked as anchored. For all the floating nodes, we check whether they have links to the anchored nodes, and calculate the new distance and the change ($\triangle$) in distance, which gives queue $\{(B, 7, 0), (C, 2, 16), (E, 10, 0)\}$ where the third value is the $\triangle$. In the next iteration, $(C, 2, 16)$ will be considered first since it has the largest $\triangle$. C will be marked as anchored. Since C has an outing edge to E and B, the new distance and $\triangle$ of E and B is calculated, which gives queue $\{(B, -4, 11), (E, -3, 13)\}$. Then E is selected, instead of only marking E as anchored, the original subtree rooted at E is considered together. Therefore the node F is selected and marked as anchored at the same time. Since E has an outing edge to D, the new distance and $\triangle$ of D is calculated, which gives queue $\{(B, -4, 11), (D, -1, 12)\}$. Since node D has the largest potential shortest distance decrease, node D is marked as anchored. In the next iteration, node B is selected. The new shortest path tree is given in the Fig. 6.

We have already known that Dijkstra's algorithm is not applicable with the networks whose link have negative weights. From the above example, when the link (A, C) change its weight to $-L(A, C)$, the correct new shortest path tree can be constructed using i-SPF. Theorem 4 indicates that this is not a special case.

Theorem 4: For any neighboring node x of c, the i-SPF algorithm can get the correct new SPT $T'_c(c, x)$.

Proof: We will use inductive reasoning to prove this theorem.
(1) We first prove the base case. When the weight of edge (c, x) is changed from $L(c, x)$ to $-L(c, x)$. The potential parent for node x is c, the potential cost for node x is $-L(c, x)$, the potential cost change for node x is $-2{\cdot}L(c, x)$. The node x will be enqueued into the Q using ENQUEUE operation. After this operation, the queue has one element in the form of $(x, (c, -L(c, x), -2{\cdot}L(c, x)))$. Because there is no node can decrease its cost by more than $-2{\cdot}L(c, x)$. Therefore, all of the descendants of node x will be placed in the correct positions during the first iteration.
(2) The inductive step is the same as in the [39], so we omitted the content. □

Theorem 5: For any node $v (v \neq c, v \neq x)$ and x is neighboring node of c, if $v \notin D(T_c, x)$ and $v \in D(T'_c(c, x), x)$, then we can get $C_x(v) < C_c(v) + C_c(x)$.

Proof: From Theorem 4, the i-SPF can get the correct new shortest path tree $T'_c(c, x)$. Assuming that $B_c(v) = y, y \neq x$ in the T_c, we have $C_c(v) = C_c(y) + C_y(v)$. Since $v \in D(T'_c(c, x), x)$, we can obtain $C'_c(v) = C_c(x) + C_x(v)$, where $C'_c(v)$ is the cost from node c to node v in the T'_c. Because the weight of link (c, x) in the $T'_c(c, x)$ is $-L(c, x)$, we can get $C'_c(v) = C_x(v) - L(c, x)(1)$. According to that $v \notin D(T_c, x)$ and $v \in D(T'_c(c, x), x)$, we can obtain $C'_c(v) < C_c(v)$ (2). Combining the (1) with (2), we have $C_x(v) < C_c(v) + L(c, x)$. Because $C_c(x) = L(c, x)$, we can get $C_x(v) < C_c(v) + C_c(x)$. □

From Theorem 5, we can see that if the shortest path from neighboring node x to d is not going through node c. The node d must be the descendant of node x in the new shortest path tree when the weight of (c, x) is changed to $-L(c, x)$. Contrarily, if the node c is included in the shortest path from neighboring node x to d. The node d must not be the descendant of node x in the new shortest path tree when the weight of (c, x) is changed to $-L(c, x)$.

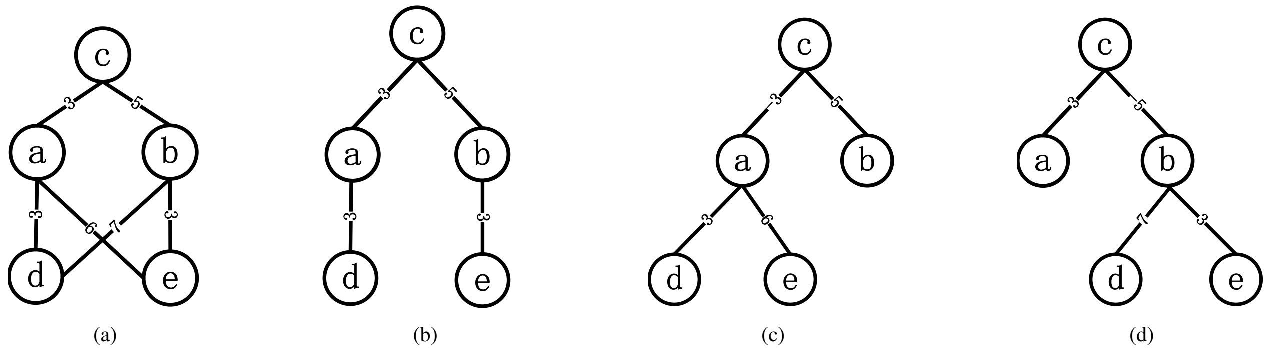


Fig. 7. An example for explaining the Theorem 5 and Theorem 6: (a) Topology, (b) T_c, (c) $T'_c(c,a)$, and (d) $T'_c(c,b)$.

Theorem 6: For any node $v(v \neq c, v \neq x)$, if $v \notin D(T_c, x)$ and $v \in D(T'_c(c,x), x)$, then we can get $N_c(v) = N_c(v) \cup x$.

Proof: From the Theorem 1 and Theorem 5, we can see that the node x is a viable backup next-hop from node c to node v, therefore we have $N_c(v) = N_c(v) \cup x$. □

We will use an example to explain the Theorem 5 and Theorem 6. Fig. 7(a) is a simple network topology. Fig. 7(b) is a shortest path tree rooted at node c, while Fig. 7(c) and 7(d) respectively represent the new SPT when the link (c,a) and (c,b) is changed to -3 and -5. Because $e \notin D(T_c, a)$ and $e \in D(T'_c(c,a), a)$, we can see that node a can be a viable backup next-hop from c to e. Due to $d \notin D(T_c, b)$, and $d \in D(T'_c(c,b), b)$, and also we can get node b can be a viable backup next-hop from c to d in the same way.

C.1 Algorithm

We will elaborate the algorithm in detail in this section. According to the above discussions, Algorithm 2 is proposed to compute the backup next-hop set which satisfies the LFC rule. The inputs of the MNP are the network topology $G = (V, E)$ and T_c, and the output is the backup next-hop set from node c to all other nodes in the network. The MNP requires several iterations. In each iteration, at the beginning, the $visited$ attribute of all nodes except c are set to $false$ (Algorithm 2 lines 2–6). We change the weight of the link (c,x) to $-L(c,x)$, and compute the weight change for (c,x) (Algorithm 2 lines 7–9), which is stored in variable $\triangle$. The shortest cost from c to v and the $visited$ attribute of node v are updated accordingly, where v is the descendant of x in the T_c (Algorithm 2 lines 10–13). For $v \in D(T_c, x)$, for each of its unvisited neighbor u, we compute the new tentative cost of u. The node u will be inserted into the Q if its new tentative cost is smaller than the old value (Algorithm 2 lines 14–23). 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 used as tie breaking. It is then added to the tree, marked as $visited$, and used as the $current\ node$ in the iteration, while its attribute like cost is also set to a permanent value (Algorithm 2 lines 25–27). The corresponding node x is added to the next-hop set $N_c(v)$ according to Theorem 6 (Algorithm 2 line 28). For each unvisited neighbor u of v, we compute the new tentative cost of u. The node u will be inserted into the Q if its new tentative cost is smaller than the old value (Algorithm 2 lines 29–36). At last, we will restore the weight of the link (c,x) (Algorithm 2 line 38).

C.2 Theoretical Analysis

In this section, we will show the performance of the algorithm MNP-e. From Theorem 7, we can see that the computational complexity of MNP-e is less than that of constructing a shortest path tree. From theorem 8, we can see that the MNP can compute all the backup next-hop set which satisfies the LFC Rule. We will describe the Theorem 7 and Theorem 8 in detail, and also their correctness is proved.

Theorem 7: The computational complexity of MNP-e is less than $O(|E|lg|V|)$ when the queue is implemented as a Heap

Proof: To compute all the backup next-hop set from node c to other nodes in the network. The MNP-e need to run k times i-SPF algorithm, where k is the number of neighbors of node c. Let N_i and M_i respectively indicate the number of nodes that must adjust their costs or parents and the number of edges which attached to these nodes when the weight of link (c,i) is changed to 0. Therefore the computational complexity of the Algorithm MNP-e is $\sum_{i=1}^{k} M_i{\cdot}lgN_i \leq \sum_{i=1}^{k} M_i{\cdot}lg|V| = O(|E|lg|V|)$. Since $N_i < |V|$, therefore the computational complexity of MNP-e is less than that of SPF. □

Theorem 8: Algorithm MNP-e can compute all the backup next-hop set that satisfies the LFC rule.

Proof: We will prove the theorem by contradiction. Supposing that there is a node $v(v \neq c, v \neq x)$, $v \notin D(T_c, x)$ and $C_x(v) < C_c(v)$, when the MNP-e is terminated. For any node $x \in N(c)$, if $C_x(v) < C_c(v)$, then $v \in D(T'_c, x)$. Therefore, according Theorem 2, we can get $x \in N_c(v)$, this contradicts the assumptions. □

V. BACKUP PATH PROTECTION ALGORITHM

A. 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.

Algorithm 2 MNP-e(G, c).

```
Input:
    Network graph $G = (V, E)$ and $T_c$
Output:
    $N_c(v), (\forall v \in V \wedge v \neq c)$
1:  for $x \in Neighbor(c)$ do
2:      for $v \in V$ do
3:          $v.visited \leftarrow false$
4:          $C'_c(v) \leftarrow C_c(v)$
5:      end for
6:      $c.visited = true$
7:      $weight \leftarrow L(c, x)$
8:      $L(c, x) \leftarrow -L(c, x)$
9:      $\triangle \leftarrow weight - L(c, x)$
10:     for $m \in D(T_c, x)$ do
11:         $C'_c(m) \leftarrow C'_c(m) - \triangle$
12:         $m.visited = true$
13:     end for
14:     for $v \in D(T_c, x)$ do
15:         for each neighbor $u$ of $v$ do
16:             if $u.visited = false$ then
17:                 $newdist \leftarrow C'_c(v) + L(v, u)$
18:                 if $newdist < C'_c(u)$ then
19:                     ENQUEUE($Q, < u, newdist >$)
20:                 end if
21:             end if
22:         end for
23:     end for
24:     while $Q$ is not empty do
25:         $< v, tc > \leftarrow$ EXTRACTMIN(Q)
26:         $v.visited \leftarrow true$
27:         $C'_c(v) \leftarrow tc$
28:         Add $x$ to $N_c(v)$
29:         for each neighbor $u$ of $v$ do
30:             if $u.visited = false$ then
31:                 $newdist \leftarrow C'_c(v) + L(v, u)$
32:                 if $newdist < C'_c(u)$ then
33:                     ENQUEUE($Q, < u, newdist >$)
34:                 end if
35:             end if
36:         end for
37:     end while
38:     $L(c, x) \leftarrow weight$
39: end for
40: return $N_c(v), (\forall v \in V \wedge v \neq c)$
```

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.

B. Algorithm

Algorithm 3 illustrates the framework of BPP. In the first step, each node in the network independently identifies its key links,

Algorithm 3 BPP.

```
Input:
    $N_c(v), (\forall v \in V \wedge v \neq c)$ obtained from Algorithm 2
Output:
    The links need to be protected by multi-hop backup method
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 KeyLinks($v$)
6:      end if
7:  end for
8:  collect KeyLinks($v$) from each $v \in V$
9:  $K = \cup_{v \in V}$ KeyLinks($v$)
10: for $e \in K$ do
11:     merge $e.P$
12:     compute $e$'s contribution $\triangle(e)$ by (6)
13: end for
14: sort $K$ in the descendent order of $\triangle(e)$
15: $K' = \emptyset$
16: compute $A(G)$
17: while $A(G) < \mathcal{R}$ and $K \neq \emptyset$ do
18:     select the first link $e$ from $K$
19:     $A(G) \leftarrow A(G) + \triangle(e)/(|V| * (|V| - 1))$
20:     add $e$ to $K'$
21:     remove $e$ from $K$
22: end while
23: distribute $K'$ to each node in the network
24: return $K'$
```

which are links that cannot be protected by MNP. [2] A key link e is also associated with the forwarding paths that go through it, denoted by $e.P$ (Algorithm 3 lines 1–7). 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, $\triangle(e)$, is computed based on $e.P$. Then the key links in K are sorted in a descendent order according to their contribution $\triangle(e)$ (Algorithm 3 lines 8–14).

The procedure of computing $\triangle(e)$ 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:

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

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 computes for a

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

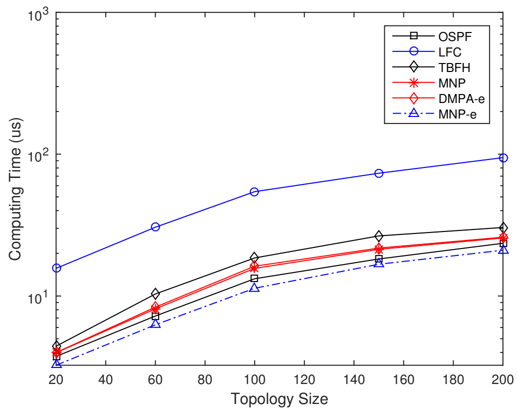


Fig. 8. Computation time on generated topologies when average node degree=4.

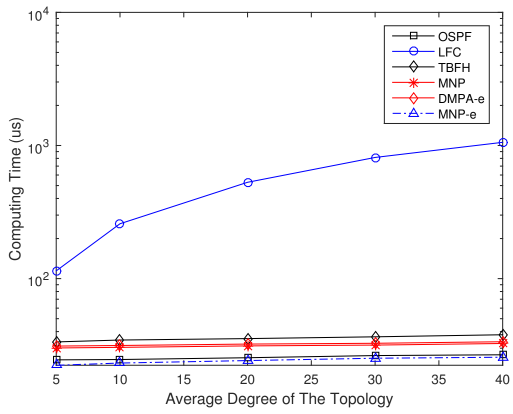


Fig. 9. Computation time on generated topologies when topology size=200.

link e is simply the new shortest path in the graph $G\backslash e$. 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 (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 (Algorithm 3 lines 15–22). 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 (Algorithm 3 line 23).

Finally, 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 (Algorithm 3 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 $\triangle(e) = \sum_{\forall(s,d)\in e.P} A^*(s,d) - A(s,d)$ 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.

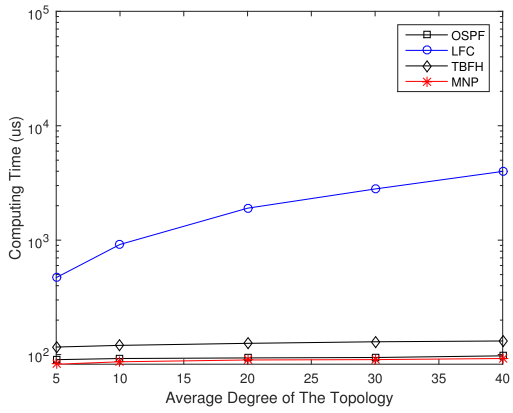


Fig. 10. Computation time on generated topologies when topology size=1000.

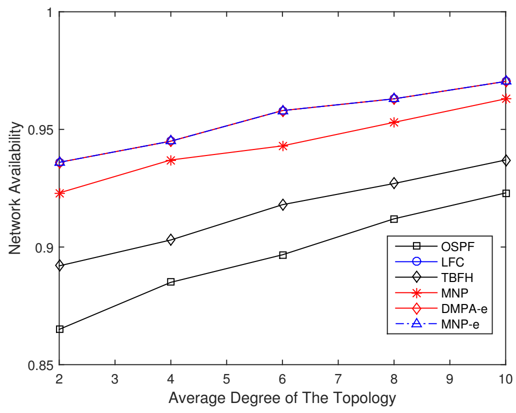


Fig. 11. Network availability on generate topologies when topology size=300.

VI. PERFORMANCE EVALUATION

As our main motivation of this paper is to achieve good computational efficiency and high network availability, the objective of our performance evaluation is twofold:

i) To indicate the effectiveness of MNP-e, we compare MNP-e with the standard OSPF and some cooperation-free mechanisms mentioned in Section II, including LFC [12], TBFH [17], MNP and DMPA-e [25]. The time complexity of other schemes mentioned in the related works are much more complex, so they are not compared here.

ii) To demonstrate the improvements achieved by HLP, which combines MNP-e and BPP, we compare HLP with the strategy that uses full protection with multi- hop backup paths. For the latter one, we use the incremental OSPF algorithm to compute a multi-hop repair path for each link. Other cooperation mechanisms mentioned in Section II not only need to modify the cur-

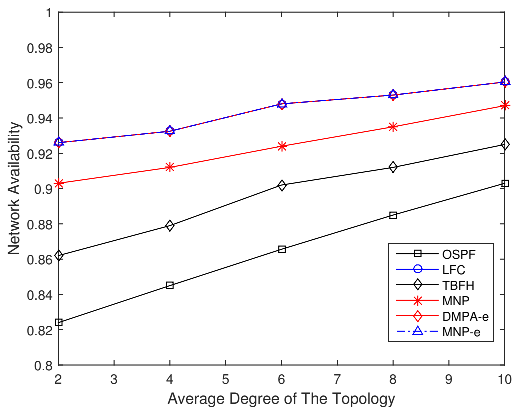


Fig. 12. Network availability on generate topologies when topology size=800.

Table 2. Parameters for BRITE.

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

rent routing protocol but also have high time complexity, so they 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 [40], 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 [41], using parameters as listed in Table 2. For simiilarity, 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 1.7 GHz and 1.5 G Memory.

A. *MNP and MNP-e*

A.1 Computation Time

The computation time of MNP, MNP-e, OSPF, LFC, and TBFH on different ISP topologies is listed in Table 3, where the number is the time averaged on all nodes in the corresponding network, and the lower, the better. It is obvious that MNP-e's performance is higher than that of OSPF, MNP, and DMPA-e's performance is close to that of OSPF. They both outperform the other two schemes, especially when the network is larger. For example, in the Sprint topology, OSPF uses 140.23 µs, MNP-e uses 137.34 µs, MNP uses 150.12 µs, DMPA-e use 153.57 µs, while TBFH and LFC's complexity are two and six times higher, respectively.

The impact of the network topology on the computing time is further illustrated in Figs. 8, 9, and 10 using synthesized topologies generated by BRITE. Figs. 8, 9, and 10 depict the corresponding computation time of each scheme , in a log-scale manner, when the topology size or the average node degree of the

Table 3. Computation time for real topologies.

	Network	Computation time (µs)					
		OSPF	LFC	TBFH	MNP	DMPA-e	MNP-e
Real	Abilene	6.82	7.27	6.97	6.83	6.87	6.52
Measured	Exodus	44.36	128.29	88.76	50.23	52.67	42.34
	Telstra	49.45	163.43	100.34	54.34	56.73	46.23
	Tiscali	79.45	398.34	183.56	83.45	85.67	75.34
	Sprint	140.23	906.78	368.12	150.12	153.57	137.34

Table 4. Network availability for real topologies.

	Network	Network availability (%)					
		OSPF	LFC	TBFH	MNP	DMPA-e	MNP-e
Real	Abilene	93.27	97.12	94.45	96.89	96.89	97.12
Measured	Exodus	84.54	91.25	86.76	90.01	90.06	91.25
	Telstra	85.34	93.23	87.34	92.11	92.57	93.23
	Tiscali	82.12	92.65	84.45	90.89	91.54	92.65
	Sprint	81.76	93.25	83.45	90.13	90.97	93.25

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 3. Figs. 9 and 10 respectively illustrate the computing time with different average node degree when the topology size is 300 and 800. We can see that the average node degree has a remarkable influence on LFC but has little influence on other four algorithms. MNP-e has the highest performance among all of the algorithms. The reason is that on each node LFC have to compute a SPT for each neighbor, while OSPF, MNP, and DMPA-e only need to compute a SPT, TBFH need to compute nearly two SPTs. The computation time of MNP-e is lower than constructing a SPT. In a large network with dense connections, i.e., when $|V| = 800$ with an average node degree of 10, LFC and TBFH may be several orders slower than MNP-e, DMPA-e, and MNP, whose performance keep close to OSPF.

A.2 Network Availability

Table 4 provides the network availability provided by each protection scheme, on the five real ISP topologies. From the results, we can see that MNP-e has an clear advantage over MNP, TBFH, DMPA-e, and OSPF, which can provide the same performance with LFC.

We also investigate how the protecting capabilities of the five schemes vary, when the topology density increases. As shown in Figs. 11 and 12, when the average node degree increases, all schemes provide better protection results, while MNP-e can provide the same performance with LFC. MNP-e and LFC are always much better than MNP, OSPF, and TBFH.

B. *HLP*

Figs. 13, 14, and 15 show the computation time of HLP, normalized by the computing time achieved by a full protection with multi-hop backup paths, on different topologies (the n in HLP-n is the average degree of the topology). The largest network availability we test is 99.99%, which is close to the theoretical maximum value, 100%. HLP has a clear advantage over full-protection. For example, HLP typically needs around 10% time of that of full protection. The advantage of HLP over the naive full-protection verifies the effect of intelligently selecting a small number of links according to their contribution to the network availability, And also the advantage of HLP over full-protection verifies the effect of taking the advantage of the much less computational intensive MNP-e.

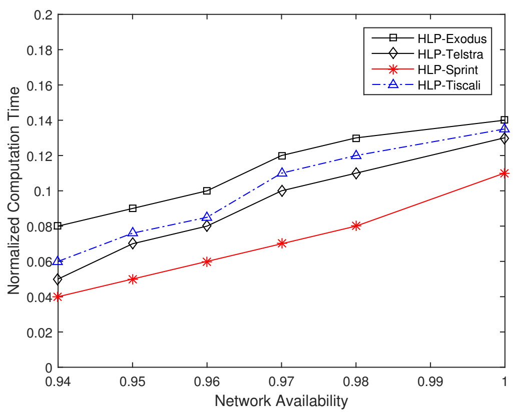


Fig. 13. Normalized computation time on real and measured topologies.

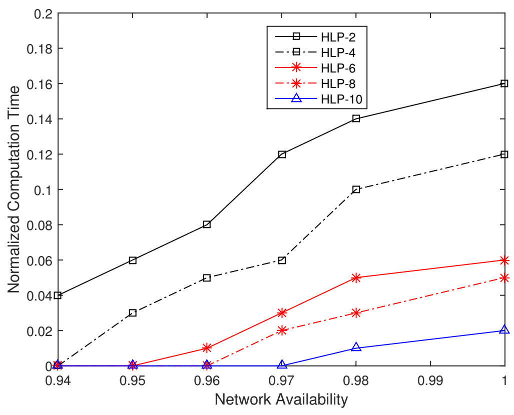


Fig. 14. Normalized computation time on generated topologies when topology size=300.

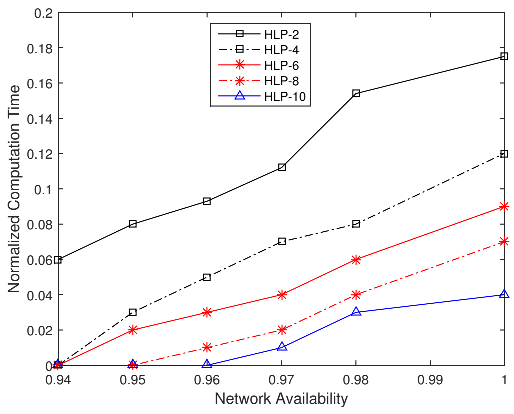


Fig. 15. Normalized computation time on generated topologies when topology size=800.

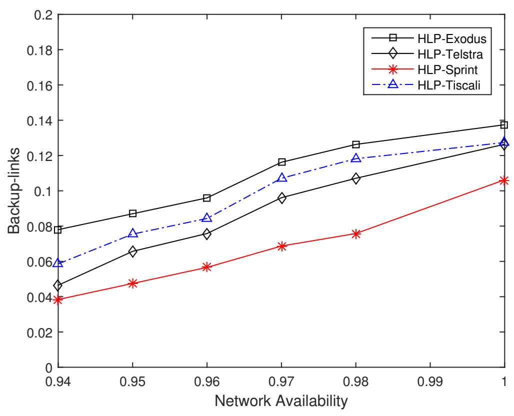


Fig. 16. Backup-links for real and measured topologies.

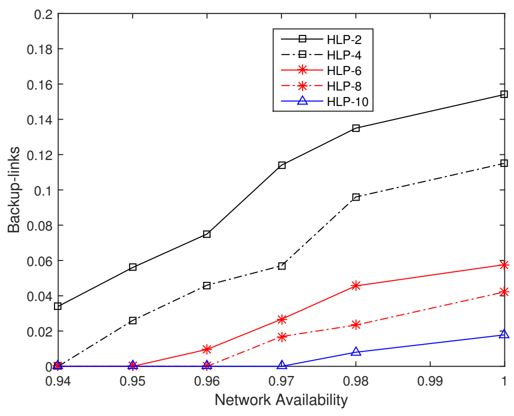


Fig. 17. Backup-links for generated topologies when topology size=300.

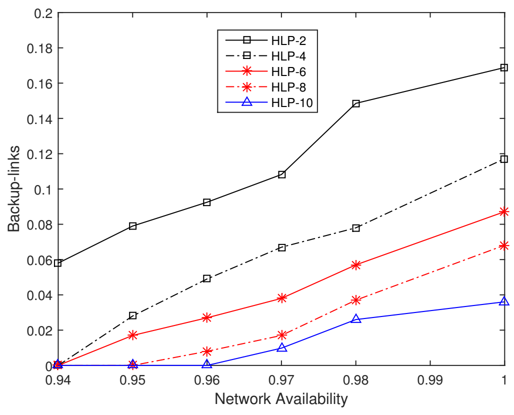


Fig. 18. Backup-links for generated topologies when topology size=800.

Figs. 16, 17, and 18 show more details. In the figures, 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-e, which effectively reduces the number of links to be protected in BPP.

VII. CONCLUSION

Traditional intra-domain routing protocols react to network failures by globally exchanging link state advertisements and re-calculating routing table. This mechanism will greatly increase the convergence time, resulting in large number of packets are discarded when network failures occur. Therefore, lots of routing protection schemes have been proposed to enhance network availability to support mission-critical and real-time applications in the internet. However, all of the aforementioned algorithms cannot strike a good balance between the implementation of efficiency and network availability. Unlike the above works, however, our main concerns are computational efficiency and network availability, as these are critical for the algorithm. Therefore, 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-e and BPP. HLP first computes multiple next-hops for source-destination pairs, then selects a minimum number of links to protect, so that it can meet network availability requirement without inducing significant overhead. Evaluation results on real and synthetic networks show that HLP can provide high network availability with low overhead. We believe networks can be made more efficient and reliable by adopting this mechanism. 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. We will also study how to deploy the HLP scheme in the hybrid software defined network architecture.

REFERENCES

[1] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, “Sentinel: Failure recovery in centralized traffic engineering,” *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1859–1872, 2019.

[2] S. Wang, H. Xu, L. Huang, X. Yang, and J. Liu, “Fast recovery for single link failure with segment routing in SDNs,” in *Proc. IEEE HPCC/SmartCity/DSS*, Aug. 2019, pp. 2013–2018.

[3] J. Bogle, *et al.,* “Teavar: Striking the right utilization-availability balance in wan traffic engineering,” in *Proc. ACM SIGCOMM*, Aug. 2019, pp. 1–15.

[4] Y. Yang, M. Xu, and Q. Li, “Fast rerouting against multi-link failures without topology constraint,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 384–397, 2018.

[5] P. Kumar, Y. Yuan, C. Yu, N. Foster, and R. Kleinberg, “Semi-oblivious traffic engineering: The road not taken,” in *Proc. USENIX NSDI*, Apr. 2018, pp. 1–15.

[6] D. Davis and V. Vokkarane, “Failure-aware protection for many-to-many routing in content centric networks,” *IEEE Trans. Netw. Sci. Eng.*, no. 99, pp. 1–16, 2019.

[7] H. Geng, X. Shi, X. Yin, Z. Wang, and S. Yin, “Algebra and algorithms for multipath qos routing in link state networks,” *J. Commun. Networks*, vol. 19, no. 2, pp. 189–200, 2017.

[8] A. Xu, J. Bi, B. Zhang, T. Xu, and J. Wu, “Ustr: A high-performance traffic engineering approach for the failed link,” in *Proc. IEEE ICDCS*, July 2018, pp. 267–277.

[9] K. J. Pai and J. M. Chang, “Dual-cists: Configuring a protection routing on some cayley networks,” *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1–12, 2019.

[10] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, “Achieving sub-second IGP convergence in large IP networks,” *ACM SIGCOMM Comput. Commun. Review*, vol. 35, no. 3, pp. 35–44, 2005.

[11] P. Francois and O. Bonaventure, “An evaluation of IP-based fast reroute techniques,” in *Proc. ACM CoNEXT*, Oct. 2005, pp. 244–245.

[12] E. A. Atlas and E. A. Zinin, “RFC 5286: Basic specification for IP fast reroute: Loop-free alternates,” *Internet RFCs*, 2008.

[13] S. B. M. Shan, “RFC 5714: IP fast reroute framework,” *Internet RFCs*, 2010.

[14] M. S. S. Bryant, S. Previdi, “Rfc 6981: A framework for IP and MPLS fast reroute usingnot-via addresses,” *Internet RFCs,* 2013.

[15] J. Moy, “Rfc 2328: OSPF Version 2,” *Internet RFCs*, 1998.

[16] X. Yang and D. Wetherall, “Source selectable path diversity via routing deflections,” in *ACM SIGCOMM Comput. Commum. Review*, vol. 36, no. 4, pp. 159–170, 2006.

[17] 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.

[18] H. Q. Vo, O. Lysne, and A. Kvalbein, “Routing with joker links for maximized robustness,” in *Proc. IEEE IFIP Netw. Conference,* May 2013, pp. 1–9.

[19] H. Q. Vo, O. Lysne, and A. Kvalbein, “Permutation routing for increased robustness in IP networks,” *International Conference on Research in Networking*, Springer, Berlin, Heidelberg, 2012, pp. 217–231.

[20] Y. Ohara, S. Imahori, and R. Van Meter, “Mara: Maximum alternative routing algorithm,” in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 298–306.

[21] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, “Fast local rerouting for handling transient link failures,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 359–372, 2007.

[22] K.-W. Kwong, L. Gao, R. Guerin, and Z.-L. Zhang, “On the feasibility and efficacy of protection routing in IP networks,” in *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1543–1556, 2011.

[23] M. Nagy, J. Tapolcai, and G. Retvari, “Node virtualization for IP level resilience,” *IEEE/ACM Trans. Netw.*, vol. PP, no. 99, pp. 1–14, 2018.

[24] M. Nagy, J. Tapolcai, and G. Retvari, “Failure-inference-based fast reroute with progressive link metric increments,” in *Proc. IEEE ICCCN*, July 2018, pp. 1–7.

[25] H. Geng, X. Shi, Z. Wang, and X. Yin, “A hop-by-hop dynamic distributed multipath routing mechanism for link state network,” *Comput. Commun.*, vol. 116, pp. 225–239, 2018.

[26] G. Apostolopoulos, “Using multiple topologies for IP-only protection against network failures: A routing performance perspective,” *ICS-FORTH, Greece, Tech. Rep*, 2006.

[27] S. Gjessing, “Implementation of two resilience mechanisms using multi topology routing and stub routers,” in *Proc. IEEE AICT-ICIW*, Feb. 2006, p. 29.

[28] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, “Path splicing,” in *Proc. ACM SIGCOMM*, Aug. 2008, pp. 27–38.

[29] S. Cho, T. Elhourani, and S. Ramasubramanian, “Independent directed acyclic graphs for resilient multipath routing,” *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 153–162, 2012.

[30] K. Lakshminarayanan, *et al.*, “Achieving convergence-free routing using failure-carrying packets,” in *Proc. ACM SIGCOMM*, Aug. 2007, pp. 241–252.

[31] F. Hao, M. Kodialam, and T. V. Lakshman, “Optimizing restoration with segment routing,” in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[32] W. Braun and M. Menth, “Loop-free alternates with loop detection for fast reroute in software-defined carrier and data center networks,” *J. Network Syst. Manag.*, vol. 24, no. 3, pp. 1–21, 2016.

[33] M. Chiesa, *et al.*, “On the resiliency of static forwarding tables,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[34] 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,” *Comput. Netw.*, vol. 54, no. 8, pp. 1300–1315, 2010.

[35] H. Geng, X. Shi, Y. Xia, Z. Wang, Z. Han, and J. Yao, “A hybrid link protection scheme for link-state routing networks,” in *Proc. IEEE IPCCC*, Dec. 2014, pp. 1–2.

[36] H. Geng, X. Shi, X. Yin, Z. Wang, and H. Zhang, “An efficient link protection scheme for link-state routing networks,” in *Proc. IEEE ICC*, June 2015, pp. 6024–6029.

[37] R. Terruggia, “Reliability analysis of probabilistic networks,” Ph.D. Dissertation, Univ. Turin, Turin, Italy, 2010.

[38] W. Feller, “An introduction to probability theory and its applications,” vol. 2, *John Wiley and Sons*, 1968.

[39] P. Narvez, K. Y. Siu, and H. Y. Tzeng, “New dynamic spt algorithm based on ball-and-string model,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 706–718, 2002.

[40] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, “Measuring ISP topologies with rocketfuel,” *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004.

[41] A. Medina, A. Lakhina, I. Matta, and J. Byers, “Brite: An approach to universal topology generation,” in *Proc. IEEE MASCOTS*, Aug. 2001, pp. 346–353.

Haijun Geng received the B.E, M.E., and Ph.D. degrees from Yantai University, Capital Normal University and Tsinghua University, in 2008, 2011 and 2015, respectively. He is now working in the School of Software Engineering, Shanxi University. His research interests include future internet architecture and largescale internet routing.

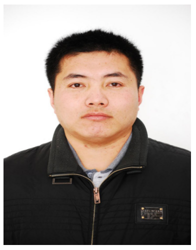
Han Zhang received the B.S. degree from Jilin University and Ph.D. from Tsinghua University. He is now working in the School of Cyber Science and Technology, Beihang University. His research concerns computer networks, network security, and AI. He is a member of IEEE.

Xingang Shi received the B.S. degree from Tsinghua University and the Ph.D. degree from the Chinese University of Hong Kong. He is now working in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include network measurement and routing protocols.

Zhiliang Wang received the B.E., M.E., and Ph.D. degrees from Tsinghua University, in 2001, 2003 and 2006, respectively. Currently he is an associate professor in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include formal methods and protocol testing, next generation internet, and network measurement.

Xia Yin received the B.E., M.E., and Ph.D. degrees from Tsinghua University in 1995, 1997, and 2000, respectively. She is a Full Professor in Department of Computer Science and Technology at Tsinghua University. Her research interests include future internet architecture, formal method, protocol testing and largescale internet routing.

Ju Zhang received the B.E. and M.E. degrees from North University of China in 2004 and 2007, respectively. He is now working in the School of Software Engineering, Shanxi University. His research interests include routing protocols, network security, and cloud computing.



Zhiguo Hu received Ph.D. degree from Tongji University, China in 2012. He is now working in the School of Computer and Information Technology, Shanxi University. His research interests include network measurement, data mining, and machine leaning.

Yong Wu received the B.E. and M.E. degrees from Taiyuan University of Technology and Guizhou University in 2005 and 2010, respectively. He is now working in the School of Software Engineering, Shanxi University. His research interests include routing protocols, network security, and cloud computing.