Efficient Computation of Loop Free Alternates

Paper ID:xxx

***Abstract*—Single network component failure may lead to heavy packet loss and degrade the network performance. The Internet Service Providers (ISPs) generally deploy Loop Free Alternates (LFA) to cope with this issue. However, the efficiency and effectiveness issues of implementing LFA are still not well addressed. The computational overhead of some existing methods (denoted by LFA) is very large, and their computational complexity is linear with the average node degree of the network. Therefore, the current methods need to consume a large amount of CPU resources, aggravate the burden on the router, and they are not suitable for practical deployment in large-scale networks. Some schemes, e.g. TBFH, have less computational overhead than LFA, however, the network availability of TBFH is lower than LFA. Therefore, to improve the routing resilience without introducing significant extra overhead, this paper propose incremental alternates computation with negative augmentation algorithm (IAC-NA) based on Incremental Shortest Path First Algorithm.**

IAC-NA first turns the problem of quickly implementation of LFA into how to efficiently calculate the minimum cost of all its neighbors to all other nodes of the network on the shortest path tree rooted at the compute node. Then a theorem for calculating the cost is presented and its correctness is proved. Finally, we theoretically analyze the time complexity of the algorithm. The computational complexity of IAC-NA is independent of the average node degree of the network.

Experiments show that compared with LFA algorithm, IAC-NA not only has less computation overhead, but also provides the same network availability as LFA.

***Index Terms*—Loop Free Alternates; distributed algorithm; Incremental Shortest Path First; network failure; network availability**

I. Introduction

With the rapid development of the Internet, more and more real-time and mission-critical applications are deployed on the internet. Since these applications are more sensitive to network delay, which impose more strict requirements on network reliability [1]. However, network failures are common in the Internet [2]. Whereas, the convergence time for the current deployed intra-domain routing protocol is the order of seconds. Therefore, network outage may be occurred even if a single network component fails. The slow convergence of the current deployed intra-domain routing protocols cannot meet the reliability requirements of real-time applications. Therefore, improving the network availability has become an urgent problem [3]–[8]. In order to improve the network availability, the routing protection method is usually adopted in the industry. The existing routing protection schemes can be 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. 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. In this paper we focus on the first type of schemes. And also, we confined our work in the link-state routing networks. Most Internet Service Providers (ISPs) prefer link-state routing instead of distance-vector routing in their intra-domain system[10], because of its merits like fast convergence and good support for metrics. Layer2 networks are also incorporating link-state routing into their network architecture, such as the standardized Transparent Interconnection of Lots of Links (TRILL) [11]. On the other hand, during topology changes caused by link or node failure, millisecond level fast convergence is preferred [12], which poses stringent performance requirement to route computation [13]. Among all the cooperation-free schemes, LFA has been favored by the industry for its simplicity and efficiency, which is to cope with the single network component failure scenario. However, the existing algorithms about LFA are time-consuming and require a large amount of router CPU resources. Therefore, this paper studies how to employ the incremental shortest path first (iSPF) algorithm to reduce the computational overhead of the LFA implementation, and proposes an efficient Intra-domain routing protection algorithm based on iSPF. In particular, our contributions can be summarized as follows:

- We propose an incremental alternates computation (IAC) algorithm based on iSPF, which can compute all the next hops satisfied DC rule.
- Theoretical analysis indicates that the computation complexity of IAC is less than that of constructing a shortest path tree and can provide the same network availability as DC.
- We propose an IAC-NA algorithm which can efficiently calculate the minimum cost of all its neighbors to all other nodes of the network on the shortest path tree rooted at the compute node. Therefore IAC-NA can completely and efficiently deal with LFA problem.
- Theoretical analysis and experiments results indicate that IAC-NA can provide the same network availability as LFA.

The rest of the paper is constructed as follows. The related work is summarized in Section II. Section III describes the network model. Section IV describes the iSPF and LFA in detail. Section V and section VI respectively present the incremental alternative computation algorithm and incremental

alternate computation with negative augmentation algorithm. In the section VII, we evaluate our algorithms on three different types of topology. And finally section VIII concludes the paper.

II. Background

Nowadays, network failures have become routine events rather than exceptions. Improving IP network resilience has drawn great attention from both industry and academia. 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 IP Fast ReRoute (IPFRR) [9]–[11], 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.

Many schemes [10], [12]–[19] that provide hop-by-hop loop-free routing are applicable in IPFRR (i.e., in the first category). Equal-Cost Multipath Routing (ECMP) [12] allows packets to be forwarded along multiple paths of equal cost, which can be specifically tuned by network operators. However, ECMP cannot offer good availability since it is limited to cases where equal cost paths exist. Loop-free Alternate (LFA) [10] proposes several basic criteria for selecting a proper next-hop, including Loop-free Criterion (LFC), Node Protection Condition (NPC) and Downstream path 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 [13] achieves faster computation by tightening DC, but still needs to construct multiple SPTs. In [20], authors propose a shortest path tree based routing protection algorithm called DMPA. DMPA guarantees loop-freeness of the induced routing path by implicitly maintaining a partial order of the routers underpinning it The time complexity of DMPA does not depend on the degree of the calculating router. However, the protection ration of DMPA is still lower than the DC. Permutation Routing [15], [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 [14], [18] 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). In [17], an interface-specific forwarding scheme called FIR is proposed, which utilizes the incoming port for backup path computation. When only single link failure notification is suppressed, a loop-free path is generated to forward packets to its destination if such a path exists. However, this design only works on the single failure case and requires a complicated pre-computation.

The second category [21]–[25] uses explicit cooperation/signaling schemes to route packets along a multi-hop repair path. Multi-topology routing [21], [22] 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 [23] 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 [24]. 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) [25] 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 [11] proposes a framework to use special not-via addresses to provide link protection by such multi-hop paths. In [26], authors propose the tunneling on demand (TOD) approach, which uses interface-specific-routing and introduces a few tunnels on demand, i.e., only for the multi-link failures that will induce routing loops to the ISR. TOD can protect the routing against arbitrary single-link failures and dual-link failures. However, TOD also need auxiliary mechanism to achieve its goal.

A few studies [27] [28] 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 [28].

Since the performance of routing and forwarding is critical to the Internet, routing protection algorithm has to be highly efficient to avoid becoming a bottleneck. However, the existing approaches often focus on finding more, or disjoint paths, rather than reducing the computation or the communication overhead, which is the focus of this paper.

Unlike the above works, however, our main concerns are computational efficiency and network availability, as these are

TABLE I: Comparison of Different Routing Protection Algorithms

Algorithm	LFC	NPC	DC
DMPA	$\times$	$-$	$\times$
TBFH	$-$	$-$	$\times$
IAC	$-$	$-$	$\surd$
IAC-NA	$\surd$	$\surd$	$\surd$

critical for the routing protection algorithm. Based on the existing work on this research area, we for the first time propose two routing protection algorithms (IAC and IAC-NA) whose complexity is less than that of Dijkstras algorithm and also have a high network availability.

A comparison of several existing solutions and ours can be seen in Table I. In Table I, the $\surd$ describes the algorithm that can compute all the next hop set satisfied the corresponding rule, the $\times$ means the algorithm can compute the partial next hop set satisfied the corresponding rule, while the $-$ represents the algorithm cannot compute the next hop set of corresponding rule. As shown in table I, only IAC-NA can be applied to three LFA rules and can compute all the next hops that meet the LFA rule. The other three algorithms do not support NPC rule. Both of the DMPA and TBFH can be used to deal with LFC and DC rule. However DMPA and TBFH cannot compute all the next hops which satisfied LFC and DC rules.

Evaluation studies [28], [29] show that LFA protects against many more failures than ECMP. Nevertheless, they also reveal that, on average, LFA offers protection against only about 90% of all possible link failures and less than 75% of node failures Despite the limited protection it achieves, LFA is already supported by some equipment vendors [30], [31]. With that in mind, Retvari et al. [32] suggested that networks should be augmented with additional links to ensure that LFA can provide recovery for all possible failures. Therefore, our algorithms can directly use the above research findings, so that they can protect all single network component failure scenarios in the network.

III. Network Model

In this paper, we will limit our research to intra-domain link state routing protocols, e.g. OSPF and IS-IS. Each router in a single routing area maintains an identical network map which allows them to compute the shortest path to every other router in a routing area. Then each router construct its FIB table employing the above information. When a packet arrives at a router, a destination address based method is using to determine how to forward the packets to its corresponding interface. When the network topology changes, the routers adjacent to the changed component detects the change and then propagates the information to its neighboring router through the link state advertisement (LSA) information. After a period of time, all routers in this routing area are aware of the change information and update their routing tables accordingly, then the network is at a stable state.

In the following sections, a network is modeled as a simple, undirected weighted graph $G = (V, E)$, where V and E respectively denote the set of nodes (routers) and the set of edges (links) in the network. Every link $(u, v) \in E$ in the network has an associated integer weight $L(u, v)$ and a failure probability $r(u, v)$. And the weights of the links in the network are symmetric. $C_c(v)$ is the lowest cost from c to v in the network. We use $N(v)$ to denote the neighbor set of the node v. For a node $u \in N(V)$, we have $C_u(v) = C_v(u) = L(u, v)$.

In a link state routing network, the computing node c builds a shortest path tree T_c rooted at itself, containing all the nodes in the network as potential destinations in the link state routing protocols, such as OSPF and IS-IS. Then the router c construct its FIB table based on the above information. In particular, we use $B_c(v)$ to represent the best/default candidate, which lies along the shortest path from c to v. Since T_c is a shortest path tree, leading to the following lemma.

Lemma 1. The Best Next-Hop Rule

$$B_c(v) = \begin{cases} v & P_c(v) = c \\ B_c(P_c(v)) & P_c(v) \neq c \end{cases} \quad (1)$$

Equation (1) in Lemma 1 means the best next-hop $B_c(v)$ for a destination v is c's direct child along the path from c to v in T_c. A shortest path routing algorithm, such as open shortest path first (OSPF) [12], [33], computes a single next-hop $B_c(v)$ by employing equation (1) at each step when a new node v is added to the SPT.

We use $G' = (V, E, (l, m, w))$ to represent the new topology when the edge $(l, m) \in E$ change its weight to w, $C_c(v, (l, m, w))$ is the shortest cost from node c to node v in the new network $G' = (V, E, (l, m, w))$. $T_c(c, x, w)$ denote the new shortest path tree when the edge $(c, x) \in T_c$ change its weight to w. For ease of reading, we summarize some symbols in the Table II.

IV. iSPF and LFA

In this section, we will discuss incremental shortest path first (iSPF) algorithm and LFA, both of which are the foundation of our work.

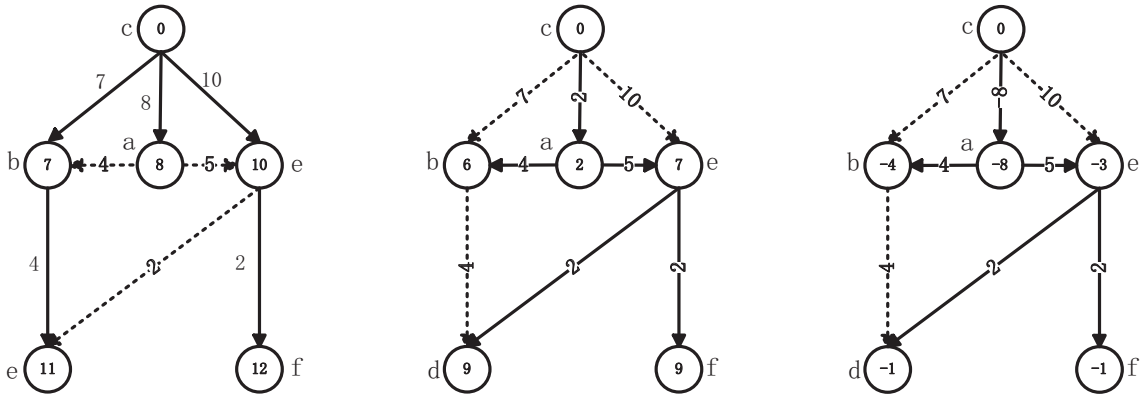
A. Incremental Shortest Path First Algorithm

The OSPF and IS-IS routing protocols which are widely deployed in todays 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 incremental shortest path first algorithm (iSPF) maintains a queue Q, each element in the queue is of the form $(n, (p, d, \delta))$, where p denotes a potential parent for node n, d denotes a potential distance for node n , and δ denotes the potential distance change for node n. The iSPF [34] is carried

TABLE II: Notations

$G=(V,E)$	Undirected graph with nodes and edges
$L(u,v)$	Direct link cost between node u and node v
$r(u,v)$	The link failure probability between node u and node v
$G'=(V,E,(l,m,w))$	The new topology when the edge $(l,m)\in E$ change its weight to w
T_c	Shortest path tree rooted at node c
$T_c(c,x,w)$	Shortest path tree rooted at node c in $G'=(V,E,(c,x,w))$
$C_c(v)$	The shortest cost from c to v in the original network$G=(V,E)$
$C_c(v,(l,m,w))$	The shortest cost from node c to node v in the new network $G'=(V,E,(l,m,w))$
$D(T_c,v)$	The descendants of node v in T_c
$N(v)$	Neighbors of node v
$D_c(v)$	Descendants of node v (itself is included) in T_c
$N_c(v)$	Next-hop set computed by node c for destination node v
$B_c(v)$	Best next-hop computed by node c for destination node v



(a) T_c (b) $T_c(c,a,2)$ (c) $T_c(c,a,-8)$

Fig. 1: An example for explaining iSPF

out as following steps:
(1) Finding all the potentially affected nodes and marking them *floating*.
(2) Computing the potential new distance, parent and the distance change δ between the old distance and the potential new distance for the potentially affected nodes.
(3) In each iteration, a node with the smallest δ (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. 1(a), 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 c, while the dotted lines are the links not in the above shortest path tree. The number inside the circle is the shortest distance from node c to that node.

At some point, the weight of the edge (c,a) is changed from 8 to 2. Nodes except c all affected nodes and are marked as *floating*. Only the root node c 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 in distance. If the change in distance is smaller than 0, the node will be enqueued into the Q. In the first iteration, the Q has only one element with the form of $\{(a,(c,2,-6))\}$. In the next iteration, $(a,(c,2,6))$ will be selected and a will be marked as *anchored*. Since a has an outing edge to e and b, the new distance and δ of b and e is calculated, which gives queue $\{(b,(a,6,-1)),(e,(a,7,-3))\}$. 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 δ of d is calculated, which gives queue $\{(b,(a,6,-1)),(d,(e,9,-2))\}$. 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. 1(b).

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 (c,a) is changed from 8 to -8. Only the root node c 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 in distance, which gives queue $\{(a,(c,2,-16))\}$. In the next iteration, $(a,(c,2,-16))$ will be selected and a will be marked as

anchored. Since a has an outing edge to e and b, the new distance and δ of e and b is calculated, which gives queue $\{(a,(b,-4,-11)),(e,(a,-3,-13))\}$. 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 δ of d is calculated, which gives queue $\{(a,(b,-4,-11)),(d,(e,-1,-12))\}$. 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. 1(c).

B. Some properties of the iSPF

Theorem 1. *Given a shortest path tree T_c, when the edge $(c,x) \in T_c$ change its weight to 0, the iSPF algorithm can get the correct new SPT $T_c(c,x,0)$.*

Proof: The proof for this theorem is exactly the same as that of [34], so the proof is omitted here. ■

Definition 1. *Given a shortest path tree T_c, we say a node d change its position in $T_c(c,x,w)$, if and only if the shortest path from c to d is different in the two trees.*

Theorem 2. *Given a shortest path tree T_c, for any node $d(d \neq c, d \neq x)$, where $x \in N(c)$*

(1) *If the position of the node d in the new SPT $T_c(c,x,0)$ is changed, then we have $C_x(d) < C_c(d)$.*
(2) *If $C_x(d) < C_c(d)$ is satisfied, then the position of the node d in the new SPT $T_c(c,x,0)$ will be changed.*

Proof: (1) If the position of the node d in the new SPT $T_c(c,x,0)$ is changed, the node d or its ancestor(s) must be enqueued into the Q using ENQUEUE operation. Assume that the difference between the old distance and the potential new distance for the node d is δ, the new shortest path cost of node d is $C_c(d)-\delta$. Since only the weight of the link (c,x) is changed to 0, we have $d \in D(T_c(c,x,0),x)$. In $T_c(c,x,0)$, we have $C_c(d,(c,x,0)) = C_c(x,(c,x,0)) + C_x(d,(c,x,0))$. Since $C_c(x,(c,x,0)) = 0$, we have $C_c(d,(c,x,0)) = C_x(d,(c,x,0))$. Because the shortest path from node x to d is not going through node c in $T_c(c,x,0)$, we have $C_x(d,(c,x,0)) = C_x(d)$. Because $C_x(d,(c,x,0)) = C_c(d)-\delta$, we can get $C_x(d) = C_c(d)-\delta$. Therefore, we have $C_x(d) < C_c(d)$.
(2) Since $C_x(d) < C_c(d)$, the shortest path from node c to node d is not via node c. When the weight of edge (c,x) is changed from $L(c,x)$ to 0, there exists a path $P = c,x,...,d$ from c to d, and the cost of which is $C_c(x)+C_x(d)$. Because $C_c(x) = 0$, the cost of path P is $C_x(d)$. Therefore, the potential distance change for node x is $C_c(d) - C_x(d) > 0$. The node x will be enqueued into the Q using ENQUEUE operation. Since only the weight of the link (c,x) is changed to 0, we have $d \in D(T_c(c,x,0),x)$. Therefore, the position of the node d in the new SPT $T_c(c,x,0)$ will be changed. ■

We have already known that Dijkstra's algorithm [35] is not applicable with the networks whose link have negative weights. From the above example, when the link (c,a) change its weight to $-L(c,a)$, the correct new shortest path tree can be constructed using iSPF. Theorem 3 indicates that this is not a special case.

Theorem 3. *Given a shortest path tree T_c, when the edge $(c,x) \in T_c$ change its weight to $-L(c,x)$, the iSPF algorithm can get the correct new SPT $T_c(c,x,-L(c,x))$.*

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 distance for node x is $-L(c,x)$, the potential distance change for node x is $-2*L(c,x)$. The node x will be enqueued into the Q using ENQUEUE operation. After this operation, the queue has one element in the form of $(x,(c,-L(c,x),-2*L(c,x)))$. Because there is no node can decrease its cost by more than $-2*L(c,x)$. 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 [34], so we omitted the content. ■

Theorem 4. *Given a shortest path tree T_c, for any node $d(d \neq c, d \neq x)$, where $x \in N(c)$*

(1) *If the position of the node d in the new SPT $T_c(c,x,-L(c,x))$ is changed, then we have $C_x(d) < C_x(c) + C_c(d)$.*
(2) *If $C_x(d) < C_x(c) + C_c(d)$ is satisfied, then the position of the node d in the new SPT $T_c(c,x,-L(c,x))$ will be changed.*

Proof: (1) If the position of the node d in the new SPT $T_c(c,x,-L(c,x))$ is changed, the node d or its ancestor(s) must be enqueued into the Q using ENQUEUE operation. Assume that the difference between the old distance and the potential new distance for the node d is δ, the new shortest path cost of node d is $C_c(d)-\delta$. Since only the weight of the link (c,x) is changed to $-L(c,x)$, we have $d \in D(T_c(c,x,-L(c,x)),x)$. In $T_c(c,x,-L(c,x))$, we have $C_c(d,(c,x,-L(c,x))) = C_c(x,(c,x,-L(c,x))) + C_x(d,(c,x,-L(c,x)))$. Since $C_c(x,(c,x,-L(c,x))) = -L(c,x)$, we have $C_c(d,(c,x,-L(c,x))) = C_x(d,(c,x,-L(c,x))) - L(c,x)$. Because the shortest path from node x to d is not going through node c in $T_c(c,x,-L(c,x))$, we have $C_x(d,(c,x,-L(c,x))) = C_x(d)$. Because $C_c(d,(c,x,-L(c,x))) = C_c(d)-\delta$, we can get $C_x(d) - L(c,x) = C_c(d) - \delta$. Therefore, $C_x(d) < L(c,x) + C_c(d)$.
(2) Since $C_x(d) < C_x(c)+C_c(d)$, the shortest path from node c to node d is not via node c. When the weight of edge (c,x) is changed from $L(c,x)$ to $-L(c,x)$, the potential distance change for node x is $C_c(d) - C_x(d) + C_x(c) > 0$. The node x will be enqueued into the Q using ENQUEUE operation. Since only the weight of the link (c,x) is changed to $-L(c,x)$, we have $d \in D(T_c(c,x,-L(c,x)),x)$. Therefore, the position of the node d in the new SPT $T_c(c,x,-L(c,x))$

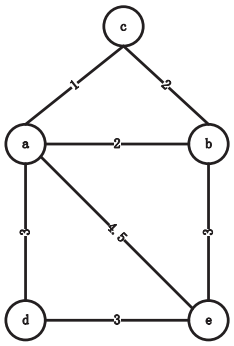


Fig. 2: An example for explaining LFA

will be changed. ■

From Theorem 4, 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 v. 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 $-L(c, x)$.

Lemma 2. *Given a shortest path tree T_c, for any node $d(d \neq c, d \neq x)$, where $x \in N(c)$, if the position of the node d in the new SPT $T_c(c, x, -L(c, x))$ is not changed, then we have $C_x(d) = C_x(c) + C_c(d).C_x(d) = C_x(c) + C_c(d)$*

Proof: From Theorem 4, we have $C_x(d) \geq C_x(c) + C_c(d)$ when the position of the node d in the new SPT $T_c(c, x, -L(c, x))$ is not changed. Because $C_x(d) \leq C_x(c) + C_c(d)$ is satisfied, we have $C_x(d) = C_x(c) + C_c(d)$. ■

C. LFA

Up to now, LFA is the only IPFRR scheme that has been deployed into commercial routers, and hence into operational IP networks. Therefore, at least two major vendors are already providing LFA out of the box, and other vendors are expected to follow suit. The LFA includes three loop-free rules. They are respectively Loop-free Criterion (LFC), Node Protection Criterion(NPC) and Downstream Path Criterion (DC). LFC can protect against a single link failure, NPC can protect against a single node failure, while DC is applicable to more complex failure scenarios. We will now dive into the details of LFA.

LFC: For packets destined to a destination v, node c $(c \neq v)$ can forward them to its any neighboring node x as long as $C_x(v) < C_c(v) + C_x(c)$, and there will be no forwarding loop in the induced forwarding path.

NPC: For packets destined to a destination v, node c $(c \neq v)$ can forward them to its any neighboring node x as long as $C_x(v) < C_x(f) + C_f(v)$, where f is the default next hop from c to v, and there will be no forwarding loop in the induced forwarding path.

DC: For packets destined to a destination v, node c $(c \neq v)$ can forward them to its any neighboring node x as long as $C_x(v) < C_c(v)$, and there will be no forwarding loop in the induced forwarding path.

The LFC rule can be interpreted as the node s is not on the shortest path from x to v, NPC shows that the node f is not on the shortest path from x to v. DC rule indicates that the neighbor x is closer to the destination v then the node s itself. From the description of the three rules, we can see that the neighbor who accords with the NPC rule must conform to the LFC rule, but the reverse is not necessarily true. A neighbor who meets the LFC rule must conform to the DC rule, but vice versa. There is no specific relationship between DC and NPC. The following is an example to explain the three rules in LFA. Fig. 2 shows a network topology consisting of 5 nodes and 6 links. As shown in Fig. 2, the default next hop from node c to node d is a, the default next hop from node c to node e is b, $C_c(d) = 4$, $C_b(d) = 5$. The node b can be used as a valid LFC next hop from the node c to the destination node d since $C_b(d) < C_b(c) + C_c(d)$. However the node b is not a valid NFC next hop from the node c to the destination node d since $C_b(d) = C_b(a) + C_a(d)$. Since $C_a(e) = 4 < C_c(e) = 5$, node a can be a valid DC next hop from c to e. Node a can also be a valid LFC next hop from c to e because $C_a(e) < C_c(e) + C_e(c)$. Node a can also be a valid NPC next hop from c to e because $C_a(e) < C_a(b) + C_b(e)$. It can be seen that the node a can be used as LFC, DC, and NPC next hop from the node c to the destination node e.

In order to achieve resilience routing, the node c can compute a candidate set of next-hops $N_c(v)$ for each destination v $(v \neq c)$ employing LFA, so that when a packet destined to v arrives at c, c can select a next-hop from $N_c(v)$ and forward this packet to.

V. Incremental Alternates Computation Algorithm

Since each node independently computes its next-hops for all destinations, in the rest of the paper, our algorithm will be described with respect to a particular node c that performs such kind of computation.

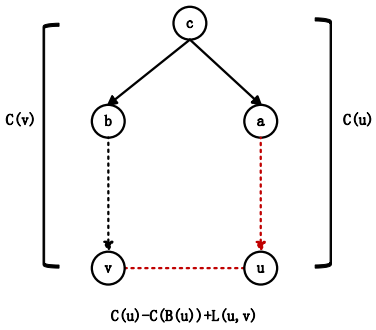


Fig. 3: Next-Hop contribution rule illustration

The DC rule is the basis of many loop-free multipath routing algorithms [13], [16], [36], which differ in their ways to find such neighboring nodes that satisfy this rule. [1] If we naively

[1]Their ways are also the root cause of their high complexity.

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. The method will involve nontrivial computational overhead. Such computation can consume a considerable amount of CPU time, preventing other critical routing functions from being executed. Thus, it is desirable to achieve this using as little CPU time as possible.

For the actual deployment on the Internet, a DC-based scheme should introduce a small additional burden on the current deployed routing protocol. This paper is dedicated to finding an efficient DC-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 DC-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 DC.***

A. Algorithm Specification

Before diving into the detail of IAC algorithm, we first describe DMPA. DMPA propose a slightly more strict rule, **The Next-Hop Contribution Rule**, than the DC rule.

Definition 2. *Given any two nodes u and v in the shortest path tree T_c, if*

$$C(u) - C(B(u)) + L(u, v) < C(v), \tag{2}$$

we say u can contribute (the best next-hop for u) to (the next-hop set for) v.

Theorem 5. (The Next-Hop Contribution Rule) *If u can contribute to v, then $C_{B(u)}(v) < C(v)$, and c can use $B(u)$, the best next hop for u, as a next hop for v without introducing forwarding loops.*

We will use an example to illustrate the Next-Hop contribution rule. In the Fig. 3, $C(u)$ denote the cost of path $p = (c, a..., u)$ in the T_c, $C(v)$ denote the cost of path $p = (c, b..., v)$ in the T_c, $C(u) - C(B(u)) + L(u, v)$ denote the cost of path $\lambda(p) \circ (u, v)$, where $\lambda(p)$ denote the subpath of p from its second node to its last node and $\circ$ is the path concatenation operator. We say u can contribute to v if $C(u) - C(B(u)) + L(u, v) < C(v)$ is satisfied. Therefore, node c can use a as a validate next hop to v.

The merit of the next-hop contribution rule is that, it is very easy to check whether inequality (2) can be satisfied for any two nodes u and v in the SPT, since all terms in inequality (2) are known at that time. So at each step when a new node v is added to the SPT, we can simply check whether any other node u added earlier can contribute to it, and add $B(u)$ to $N(v)$ if that is true. Similarly, if v can contribute to u, just add $B(v)$ to $N(u)$. In particular, we only need to do this test if u and v are neighbors in the network, since otherwise $L(u, v) = \infty$ and (2) can never be satisfied. With this rule, we can compute $N(v)$ for any node v in a way faster than other multipath algorithms, without introducing loops.

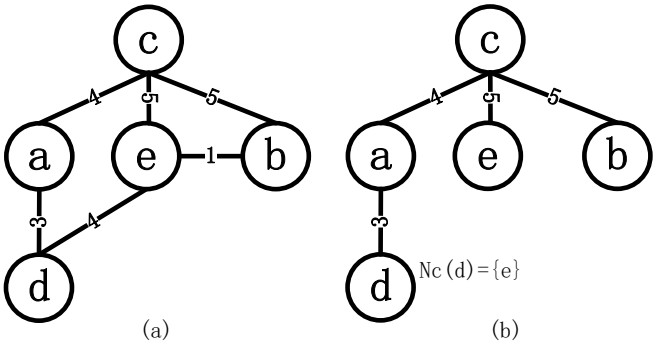


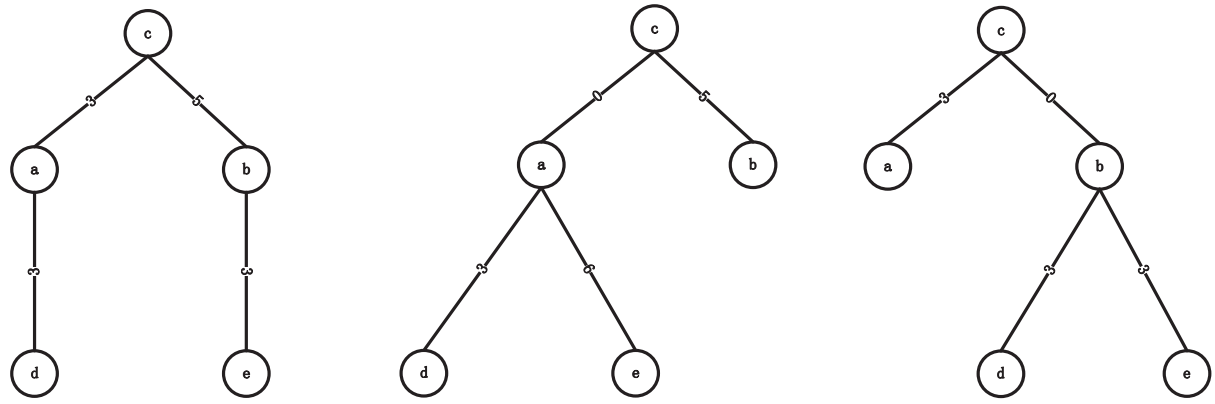
Fig. 4: An example for explaining the drawback of DMPA

Fig. 4 gives an example to explain the drawback of DMPA. Fig. 4(a) is a simple network topology which consists of 5 node and 6 edges. Fig. 4(b) is a shortest path tree rooted at node c. From Fig. 4(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 DMPA. 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 DMPA does not depend on the degree of the calculating router. However, the network availability of DMPA is lower than that of the DC. Unlike the above work, DMPA, however, our main concerns are computational efficiency and network availability, as these are critical for the Algorithm. 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 DC. We first provide two theorems before formally describing the details of the algorithm. The following two theorems describe how to compute all the DC backup next-hop set for node c. And the computation overhead can be dramatically reduced via reducing the times of the operation.

Theorem 6. *Given a shortest path tree T_c, for any node $d(d \neq c, d \neq x)$, if $d \notin D(T_c, x)$ and $v \in D(T_c(c, x, 0), x)$, then we can get $N_c(d) = N_c(d) \cup x$.*

Proof: From the Theorem 1, Theorem 2 and DC rule, we can see that the node x is a viable backup next-hop from node c to node d, therefore we have $N_c(d) = N_c(d) \cup x$. ■

We will use a simple example to explain the Theorem 6. Fig. 5 is a simple network work consists of 5 routers and 6 edges. Fig. 6(a) is a shortest path tree rooted at node c, while Fig. 6(b) and 6(c) respectively represent the new SPT when the weight of the links (c, a) and (c, b) is changed to 0. Because $e \notin D(T_c, a)$ and $e \in D(T'_c, a)$, 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 $d \in D(T'_c, b)$, node b can be a viable backup next-hop from c to d, and also we can get ode b can be a viable backup next-hop from c to g in the same way.



(a) T_c (b) $T_c(c, a, 0)$ (c) $T_c(c, b, 0)$

Fig. 6: An example for explaining the Theorem 6

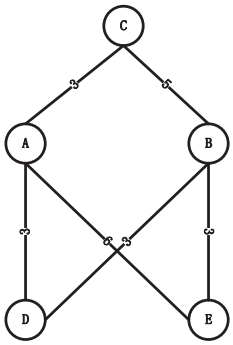


Fig. 5: network topology

B. Algorithm

We are now in a position to describe the IAC algorithm. According to the above discussions, algorithm IAC is proposed to compute the backup next-hop set which satisfies the DC rule. The inputs of the IAC 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 IAC requires several iterations. In each iteration, at the beginning, the $visited$ attribute of all nodes except c are set to $false$ (lines 2 $\sim$ 5). We change the weight of the link (c, x) to 0, and compute the weight change for (c, x) (lines 6 $\sim$ 8), which is stored in variable $\triangle$. The shortest cost from c to v and the $visited$ attribute of node v are updated accordingly (lines 9 $\sim$ 11). 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 (lines 12 $\sim$ 17). 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 (lines 19 $\sim$ 21). The corresponding node x is added to the next-hop set $N_c(v)$ according to Theorem 6 (line 22). 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 (lines 23 $\sim$ 27). At last, we will restore the weight of the link (c, x) (line 28).

C. Theoretical Analysis

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

Theorem 7. *The computational complexity of IAC 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 IAC need to run the iSPF algorithm k times, 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 anchored to these nodes when the weight of link (c, i) is changed to 0. Therefore the computational complexity of the Algorithm INC is $\sum_{i=1}^{k} M_i * lgN_i \leq \sum_{i=1}^{k} M_i * lg|V| = O(|E|lg|V|)$. Since $N_i < |V|$, therefore the computational complexity of IAC is less than that of SPF. ■

Theorem 8. *Algorithm IAC can compute all the backup next-hop set that satisfies the DC Rule.*

Proof: We will prove the theorem by contradiction. Supposing that there is a node $d(d \neq c, d \neq x)$, $d \notin D(T_c, x)$ and $C_x(d) < C_c(d)$, when the IAC is terminated. For any node $x \in N(c)$, if $C_x(d) < C_c(d)$, we have $d \in D(T_c, x)$ according to Theorem 1 and Theorem 2. This contradicts the assumption. ■

Algorithm 1: IAC

Input: Network graph $G = (V, E)$
Input: Node c running the algorithm
Output: $N_c(v), (\forall v \in V \wedge v \neq c)$

```
1  for x ∈ N(c) do
2      for v ∈ V do
3          v.visited ← false;
4          C'_c(v) ← C_c(v);
5      c.visited = true;
6      weight ← L(c, x);
7      L(c, x) ← 0;
8      △ ← weight − L(c, x);
9      for m ∈ D(T_c, x) do
10         C'_c(m) ← C'_c(m) − △;
11         m.visited = true;
12     for v ∈ D(T_c, x) do
13         for each neighbor u of v do
14             if u.visited = false then
15                 newdist ← C'_c(v) + L(v, u);
16                 if newdist < C'_c(u) then
17                     δ ← newdist − C'_c(u)
                       ENQUEUE(Q, (u, (v, newdist, δ)));
18     while Q is not empty do
19         < v, tc > ← EXTRACTMIN(Q);
20         v.visited ← true;
21         C'_c(v) ← tc;
22         Add x to N_c(v);
23         for each neighbor u of v do
24             if u.visited = false then
25                 newdist ← C'_c(v) + L(v, u);
26                 if newdist < C'_c(u) then
27                     ENQUEUE(Q, (u, (v, newdist, δ)));
28     L(c, x) ← weight;
29 return N_c(v), (∀v ∈ V ∧ v ≠ c)
```

D. Discussions

IAC algorithm can only handle DC rule. However, it cannot be directly applied to LFC and NPC rules. Therefore, IAC cannot completely solve LFA problem. We want to ask whether we can find an general algorithm that can completely and efficiently solve LFA. We will discuss this issue in the following sections.

VI. Incremental Alternate Computation with Negative Augmentation

For any node $x \in N(c)$, if we can quickly calculate $C_x(v), v \in V$ in the T_c, then a efficient LFA-based method can be achieved. Therefore, the problem that needs to be solved in this paper can be described as follows: For any node $x \in N(c)$, if T_c it is given, whether could we find an efficient algorithm which can quickly compute $C_x(v), v \in V$ or not. Theorem 9 answers how to solve the above problem and gives proof.

Theorem 9. *Given a computing node c and T_c, for any node $d(d \neq c, d \neq x)$ where $x \in N(c)$,*

(1) *if $d \in D(T_c, x)$, then we have $C_x(d) = C_c(d) - C_c(x)$.*
(2) *if $d \notin D(T_c, x)$ and $d \in D(T_c, (c, x, -L(c, x)))$, then we have $C_x(d) = C_c(d, (c, x, -L(c, x))) + C_c(x)$.*
(3) *if $d \notin D(T_c, x)$ and $d \notin D(T_c, (c, x, -L(c, x)))$, then we have $C_x(d) = C_c(d) + C_c(x)$.*

Proof: (1)Since $v \in D(T_c, x)$, we have $C_c(v) = C_c(x) + C_x(v)$. Therefore, we can easy get $C_x(v) = C_c(v) - C_c(x)$.
(2)Since $v \in D(T_{c,x,-L(c,x)}, x)$, we have $C_c(d, (c, x, -L(c, x)) = C_c(x, (c, x, -L(c, x)) + C_x(d, (c, x, -L(c, x)))$. From Theorem 2, we can see that $C_x(d, (c, x, -L(c, x))) = C_x(d)$. Therefore, we have $C_c(d, (c, x, -L(c, x))) = C_c(x, (c, x, -L(c, x))) + C_x(d)$. Because $C_c(x, (c, x, -L(c, x))) = -C_c(x)$, we have $C_x(d) = C_c(d, (c, x, -L(c, x))) + C_c(x)$.
(3)From lemma 2, we can get $C_x(d) = C_c(d) + C_c(x)$. ■

We will employ a simple example to explain the Theorem 9. Fig. 7(a) shows a network topology consisting of 5 nodes and 6 edges. Fig. 7(b) shows the shortest path tree T_c rooted at the node c. Figure 7(c)shows the shortest path tree constructed using iSPF when the weight of link (c, a) is changed to -3. We have $d \in D(T_c, a)$ from Fig. 7(b) , so $C_a(d) = C_c(d) - C_c(a) = 3$. Because $e \notin D(T_c, a)$ and $e \in D(T_c(c, a), a)$, we have $C_a(e) = C_c(e, (c, a)) + C_c(a) = 6$ from Theorem 9. Since $b \notin D(T_c, a)$ and $b \notin D(T_c(c, b), a)$, we have $C_a(b) = C_a(c) + C_c(b) = 8$ from Theorem 9.

A. IAC-NC

Since the execution process of algorithm INC and INC-NA is basically similar, we will not repeat the same part in both of the algorithms. However, for the completeness of the description, we still lists the pseudo code of the algorithm INC-NA. Below, we will list the differences between the two algorithms. The line 7 in INC-NA is changed into $L(c, x) = -L(c, x)$. The line 22 in INC-NA is changed into Compute $C_x(v)$ using Theorem 9. At last lines 29-32 are added to compute LFA next hop set for node c.

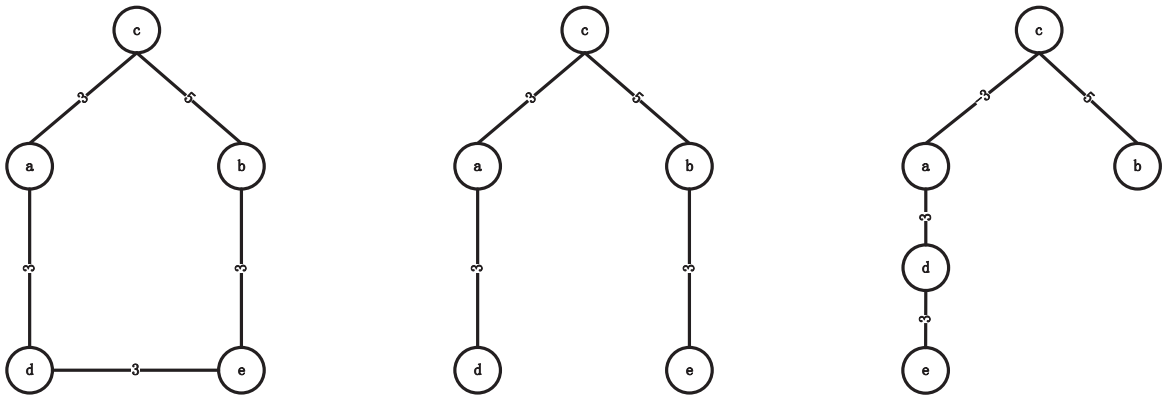
B. Theoretical Analysis

Theorem 10. *The computational complexity of IAC-NA is less than $O(|E|lg|V|)$ when the queue is implemented as a Heap.*

The proof for Theorem 10 is similar with that of Theorem 7, so we omit it.

Theorem 11. *Algorithm IAC-NA can compute all the next hop set for node c that satisfies the LFA Rule.*

Proof: For any node $x \in N(c)$, the $C_x(v), v \in V$ will be calculated when the lines (1-28) of the IAC-NA are executed. As long as we know the above values, we can use the LFA rule to compute all the LFA next hop for node c. Therefore, the theorem is established. ■



(a) network topology (b) T_c (c) $T_c(c, a, -3)$

Fig. 7: An example for explaining Theorem 9

VII. Performance Evaluation

To evaluate our algorithm, we use the real network Abilene (11 nodes, 14 links), and five ISP topologies which are measured by Rocketfuel [37], including AS3967 (79 nodes, 147 links), AS1221 (108 nodes, 153 links), AS3257 (161 nodes, 328 links), AS6461 (128 nodes, 372 links) and AS1239 (315 nodes, 972 links). We also generate synthetic topologies with BRITE [38], using parameters as listed in Table III. 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, 1.7GHz and 1.5G Memory. Because neither DMPA nor TBFH can implement the NPC rule, IAC can only implement DC rule, so in the experiment we only compare the performance of the three methods on achieving DC rule.

A. Computation complexity

Theoretical analysis has indicated that the time complexity of IAC/IAC-NA is less than that of building of a shortest path tree, which has a great advantage over DC, TBFH and DMPA. In order to further verify the computational performance, we make simulations on different topologies. In this section, we evaluate the computational overhead of different algorithms. In order to avoid the uncertain factors impact the algorithms performance. The computational overhead of an algorithm is defined as the ratio of computation time of the algorithm to that of SPF (shortest path first).

Fig. 8 indicates the computational overhead obtained by different algorithms on Abilene and Rocketfuel topologies. From the Fig. 8, we can see that IAC/IAC-NA has the lowest computation overhead among all the algorithms. The computation overhead of IAC/IAC-NA is less than computing a SPT, while DMPA need to construct a SPT and TBFH need to compute two SPTs. The computation overhead of DC is proportionally to the degree of the network average node degree.

Fig. 9 illustrates the relationship between the computation overhead and topology size on generated topologies when the average node degree is 6. As Fig. 9 shows, the computation overhead of all the algorithms does not depend on the topology size. The computation overhead of IAC/IAC-NA is lowest among all the algorithms.

Fig. 10 presents the relationship between the computation overhead and average node degree on generated topologies

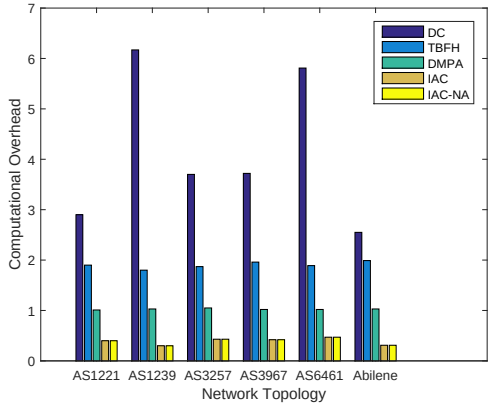


Fig. 8: Computation overhead on real and measured topologies

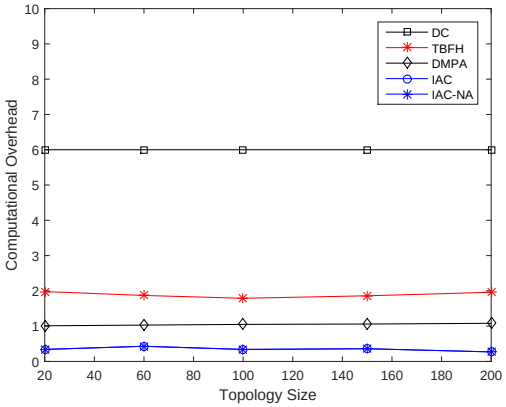


Fig. 9: Computation overhead on generated topologies when average node degree is 6

Algorithm 2: IAC-NA

Input: Network graph $G = (V, E)$
Input: T_c
Output: $N_c(v), (\forall v \in V \wedge v \neq c)$

for $x \in N(c)$ **do**
  for $v \in V$ **do**
    $v.visited \leftarrow false$;
    $C'_c(v) \leftarrow C_c(v)$;
  $c.visited = true$;
  $weight \leftarrow L(c, x)$;
  $L(c, x) \leftarrow -L(c, x)$;
  $\triangle \leftarrow weight - L(c, x)$;
  for $m \in D(T_c, x)$ **do**
    $C'_c(m) \leftarrow C'_c(m) - \triangle$;
    $m.visited = true$;
  for $v \in D(T_c, x)$ **do**
    for *each neighbor* u *of* v **do**
      if $u.visited = false$ **then**
        $newdist \leftarrow C'_c(v) + L(v, u)$;
        if $newdist < C'_c(u)$ **then**
          ENQUEUE$(Q, (u, (v, newdist, \delta))$;
  while Q *is not empty* **do**
    $< v, tc >\leftarrow$ EXTRACTMIN(Q);
    $v.visited \leftarrow true$;
    $C'_c(v) \leftarrow tc$;
    Compute $C_x(v)$ using Theorem 9;
    for *each neighbor* u *of* v **do**
      if $u.visited = false$ **then**
        $newdist \leftarrow C'_c(v) + L(v, u)$;
        if $newdist < C'_c(u)$ **then**
          $\delta \leftarrow newdist - C'_c(u)$
          ENQUEUE$(Q, (u, (v, newdist, \delta))$;
  $L(c, x) \leftarrow weight$;
for $d \in V, d \neq c$ **do**
  for $x \in N(c)$ **do**
    if *LFA is satisfied* **then**
      Add x to $N_c(d)$;
return $N_c(v), (\forall v \in V \wedge v \neq c)$

when the topology size is 200. As the average node degree increases, the computation overhead of DC increases accordingly. And also IAC/IAC-NA has the highest performance among all the algorithms. Therefore, the above experiment results are consistent with the theoretical analysis described above.

B. Protection ratio

In this section, we will use protection ratio to measure path diversity of different algorithms. The protection ratio is defined as

$$Pr = \frac{\sum_{c,d \in V, c \neq d} K(c, d)}{|V| * (|V| - 1)} \tag{3}$$

$$K(c, d) = \begin{cases} 1 & |N_c(d)| \geq 2 \\ 0 & |N_c(d)| = 1 \end{cases} \tag{4}$$

It can be seen from this definition that the higher protection ratio, the higher the path diversity. Fig. 12 depicts the protection ratio obtained by different algorithms on Abilene and Rocketfuel topologies. Fig. 13 presents the relationship between the protection ratio and average node degree on generated topologies when the topology size is 200. From Fig. 12 and Fig. 13, we can see that INC, INC-NA and DC have the similar protection ratio, which have better performance than both of DMPA and TBFH. Since protection ratio cannot accurately describe end-to-end network availability. Therefore, we will use network availability to measure the end-to-end availability of different algorithms in the next section.

TABLE III: Parameters for BRITE

Model	N	HS	LS
Waxman	20-1000	1000	100
m	NodePlacement	GrowthTypem	*alpha*
2-40	Random	Incremental	0.5
beta	BWDist	BwMin	BwMax
0.5	Heavy Tail	100.0	1024.0

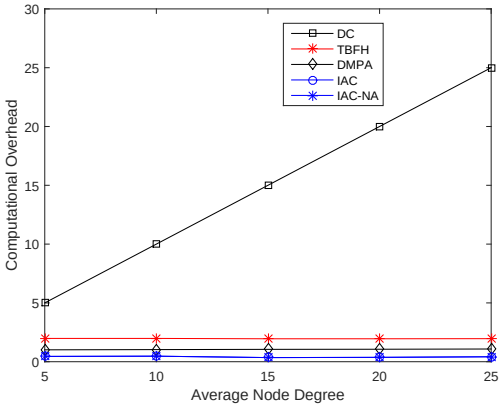


Fig. 10: Computation overhead on generated topologies when topology size is 200

C. Network availability

We formally define the network availability $A(G)$ as follows (similar to that in [39]), 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

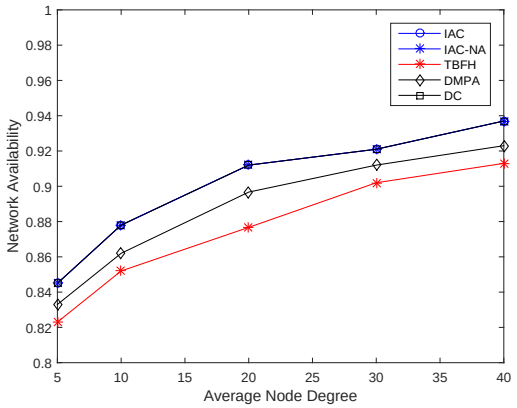


Fig. 11: Network Availability on Generated Topologies When Topology Size=200

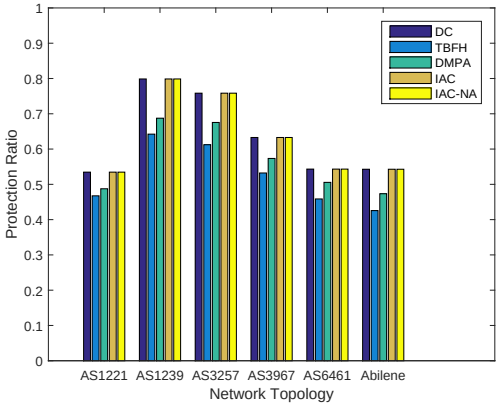


Fig. 12: Protection ratio on real and measured topologies

$p_i(s,d)$. 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:

$$P(A_i) = \prod_{\forall(m,n)\in P_i(s,d)} r(m,n) \tag{5}$$

According to the Inclusion-Exclusion principle [40], the end-to-end availability of a source-destination pair can be expressed as:

$$A(s,d) = \sum_{k=1}^{n} (-1)^{(k-1)} S_k \tag{6}$$

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:

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

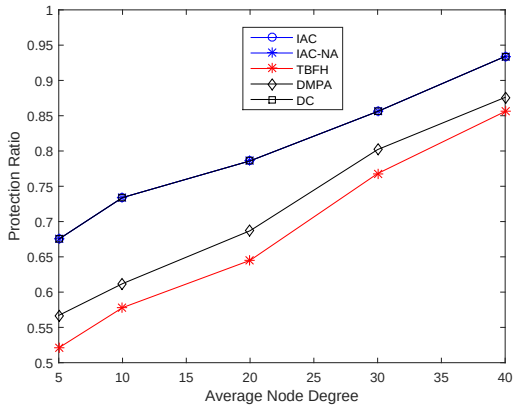


Fig. 13: Protection ratio on generated topologies when topology size is 200

TABLE IV: Network Availability for Real and Measured Topologies

	Network	Network Availability(%)			
		IAC/IAC-NA	DC	TBFH	DMPA
Real	Abilene	97.12	97.12	94.45	96.89
Measured	AS3967	91.25	91.25	86.76	90.01
	AS1221	93.23	93.23	87.34	92.11
	AS3257	92.65	92.65	84.45	90.89
	AS1239	93.25	93.25	83.45	90.13

Then, the network availability can be computed as

$$A(G) = \frac{\sum_{s,d\in V, s\neq d} A(s,d)}{|V| * (|V|-1|)} \tag{7}$$

Table IV provides the network availability provided by each protection scheme, on the real and measured topologies. From the results, we can see that IAC/IAC-NA has a clear advantage over TBFH and DMPA, and has the same performance as DC.

Fig. 11 illustrates the relationship between the network availability and the average node degree. As Fig. 11 shows, as the average node degree increase, the network availability increases too. Note that when the average node degree increases, all schemes provide better protection results, while IAC/IAC-NA and DC are always much better than DMPA and TBFH.

VIII. Conclusion

This paper is mainly focusing on the problem that how to efficiently and effectively deploy LFA on ISPS' networks. To this end, an incremental alternates computation with negative augmentation algorithm (IAC-NA) based on Incremental Shortest Path First Algorithm was proposed. Unlike majority of schemes proposed in the literature, IAC-NA does not need to construct many SPTs on a node, which would entail significant computation overhead. We also established the correctness of our scheme, together with some other important properties. On top of that, simulation results using several realistic network instances gave evidence that IAC-NA is very

competitive in terms of network availability as well. The simulation results showed that IAC-NA can achieve the same network availability as the state-of-the-art LFA with small computation overhead.

In the future, we will focus on studying the link failure characteristics. We believe that accurate link failure model will accurately improve the performance evaluation. We will use virtualization technology to deploy our algorithms on CERNET2 [41], which is the China Education and Research network.

IX. Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (Grant No. 61702315).

References

[1] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, "We've got you covered: Failure recovery with backup tunnels in traffic engineering," in *IEEE 24rd International Conference on Network Protocols (ICNP)*, 2016, pp. 37–385.

[2] 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.

[3] P. Krist, "Scalable and efficient multipath routing: Complexity and algorithms," in *IEEE 23rd International Conference on Network Protocols (ICNP)*, 2015, pp. 376–385.

[4] J. Yallouz, O. Rottenstreich, P. Babarczi, A. Mendelson, and A. Orda, "Optimal link-disjoint node-somewhat disjoint paths," in *IEEE 24rd International Conference on Network Protocols (ICNP)*, 2016, pp. 1–10.

[5] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep forwarding: Towards k-link failure resilient routing," *Proceedings - IEEE INFOCOM*, pp. 1617–1625, 2014.

[6] T. Elhourani, A. Gopalan, S. Ramasubramanian, T. Elhourani, A. Gopalan, and S. Ramasubramanian, "Ip fast rerouting for multi-link failures," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3014–3025, 2016.

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Usenix Conference on Networked Systems Design and Implementation*, 2013, pp. 113–126.

[8] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Symposium on Sdn Research*, 2016, p. 9.

[9] S. B. M. Shan, "Ip fast reroute framework, rfc 5714," 2010.

[10] E. A. Atlas and E. A. Zinin, "Basic specification for ip fast reroute: Loop-free alternates, rfc 5286," 2008.

[11] M. S. S. Bryant, S. Previdi, "A framework for ip and mpls fast reroute using not-via addresses, rfc 6981," 2013.

[12] J. Moy, "Rfc 2328: Ospf version 2," *Internet Society (ISOC)*, 1998.

[13] 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.

[14] Y. Ohara, S. Imahori, and R. Van Meter, "Mara: Maximum alternative routing algorithm," in *Proceedings - IEEE INFOCOM*, 2009, pp. 298 – 306.

[15] H. Q. Vo, O. Lysne, and A. Kvalbein, "Routing with joker links for maximized robustness," in *IFIP Networking Conference, 2013*. IEEE, 2013, pp. 1–9.

[16] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections." in *SIGCOMM*. ACM, 2006, pp. 159–170.

[17] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast local rerouting for handling transient link failures," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 359 – 372, 2007.

[18] 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 Transactions on Networking*, vol. 19, no. 5, 2011, pp. 1543 – 1556.

[19] H. Q. Vo, O. Lysne, and A. Kvalbein, "Permutation routing for increased robustness in ip networks," in *IFIP Networking Conference, 2012*, 2012, pp. 217 – 231.

[20] H. Geng, X. Shi, X. Yin, and Z. Wang, "Dynamic distributed algorithm for computing multiple next-hops on a tree," in *21th IEEE International Conference on Networking ICNP 2013*. IEEE, 2013.

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

[22] S. Gjessing, "Implementation of two resilience mechanisms using multi topology routing and stub routers," in *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*. IEEE, 2006, pp. 29–29.

[23] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," in *SIGCOMM*, 2008, pp. 27–38.

[24] 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.

[25] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *ACM SIGCOMM 2007: Conference on Computer Communications*, Kyoto, Japan, 2007, pp. 241 – 252.

[26] Y. Yang, M. Xu, and Q. Li, "Fast rerouting against multi-link failures without topology constraint," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 384–397, 2018.

[27] 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.

[28] 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.

[29] M. Gjoka, V. Ram, and X. Yang, "Evaluation of ip fast reroute proposals," in *International Conference on Communication Systems Software and MIDDLEWARE*, 2007, pp. 1–8.

[30] S. J. Cisco Systems, "Cisco ios xr routing configuration guide," 2008.

[31] S. Juniper Networks, "Junos os routing protocols configuration guide," 2009.

[32] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar, "Ip fast reroute: Loop free alternates revisited," *Proceedings - IEEE INFOCOM*, vol. 2, no. 3, pp. 2948–2956, 2011.

[33] J. T. Moy, *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[34] P. Narvez, K. Y. Siu, and H. Y. Tzeng, "New dynamic spt algorithm based on ball-and-string model," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 706–718, 2002.

[35] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[36] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Efficient algorithms for multi-path link-state routing," in *ISCOM'99*, 1999.

[37] 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.

[38] 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.

[39] R. Terruggia, "Reliability analysis of probabilistic networks," *PhD Thesis*, 2010.

[40] W. Feller, "An introduction to probability theory and its applications vol. i," *Wiley*, 1968.

[41] "http://www.cernet2.net/."