Networking 2013 1569696291

Routing with Joker Links for Maximized Robustness

Hung Quoc Vo
Simula Research Laboratory

Olav Lysne
Simula Research Laboratory

Amund Kvalbein
Simula Research Laboratory

***Abstract*—IP Fast Reroute methods that are currently deployed in link-state routing protocols with hop-by-hop forwarding, such as Equal-Cost Multi-Path (ECMP) and Loop Free Alternates (LFA), share two common important properties. First, they never form forwarding loops, even when there are multiple independent failures in the network. Second, they do not introduce non-standard packet marking to convey information associated with network faults. However, these Fast Reroute methods give very poor fault coverage; in most cases below 50% of links are protected when using typical link weight settings. This paper presents a new routing method that combines the concept of permutation routings with joker links, called joker-capable permutation routings. Our method results in a routing strategy that shares with ECMP and LFA the stated important properties. Through experiments we show that we protect more than 95% of links for all tested ISP networks. Measurements also show that our method is computationally feasible and performs traffic distribution efficiently under high fluctuations of traffic demands.**

I. Introduction

The increasing use of the Internet to carry time-sensitive applications makes it more important that networks are robust against even short-lived failures. Measurements indicate that unplanned failures account for 70% of all outages in networks, and many of them are short-lived [1]. For that reason, fast recovery from network failures becomes an important goal in recent routing solutions. Currently implemented IP Fast Reroute (IPFRR) methods for intra-domain networks with traditional (non-MPLS) link state routing protocols are Equal-Cost Multi-Path (ECMP) and Loop Free Alternates (LFA), both of which share two common important properties. First, they are free of loops, even when networks suffer from multiple uncorrelated failures. Second, they do not introduce non-standard packet marking in the IP header to identify network failures. However, both ECMP and LFA give very poor coverage for single link faults; in most cases they provide below 50% of nodes with fast reroute capability [2].

Provision of full coverage for failures has become the main routing objective in recent proposed routing schemes. Many solutions promise 100% coverage for single link faults, e.g. MRC [3], FIFR [4], [5], IDAG [6], and ESCAP [7], while others seek to offer 100% coverage for dual link faults, e.g. 2DLMRC [8], ESCAP-DL [9], and LOLS [10]. They, however, relax one of two mentioned design properties for adoption. Methods like [4], [5], [7], and [9] may cause forwarding loops under multiple unrelated failures while those like [3], [6], [8], and [10] need special marking for rerouted packets.

In this paper, we propose a new IPFRR method that combines the concept of permutation routings, first introduced in [11], with joker links, proposed in O2 routing [12]. Permutation routing [11] is a flexible approach for calculating multiple loop-free next-hops in networks with the traditional hop-by-hop forwarding. Permutation routing is based on the fact that any loop-free routing strategy can be expressed as a *permutation* (sequence) of nodes that are involved in traffic forwarding to a destination. Joker links in a routing are bi-directional forwarding links, and two nodes connecting through a joker link can use each other as backup next-hops. Joker links can help increase the network robustness since the restriction that all forwarding should go in only one direction inevitably leaves a significant number of nodes with only one routing choice. The combination of permutation routing and the joker links, which we call joker-capable permutation routing, aims to *maximize* fast rerouting coverage for IP networks and work with existing link state routing protocols, e.g. OSPF or IS-IS, by sharing with ECMP and LFA the two listed properties.

The main focus of this paper is to define robustness objectives for joker-capable permutation routings and provide a general joker link concept, from which all possible joker links can be identified during the course of constructing ordering of nodes in the permutation. The resulting routing is proven to *maximize* the number of source-destination (S-D) pairs with *at least* two available next-hops and completely be free of loops under multiple uncorrelated failures. The formation of our routing method is solely based on the topology information that is collected by a link state routing protocol, and hence no new control plane signaling is needed.

By maximizing number of S-D pairs with at least two next-hops, our routings result in significantly increased multipath capability for each S-D pair. This gives increased robustness, but raises concerns on traffic congestion and high path inflation. Through extensive flow-based simulations, we show that our routing approaches together with Distributed Exponentially-weighted Flow Splitting (DEFT) [13], a flexible load-balancing mechanism for unequal-cost multipath, provide low average link utilization and limited average path length that can be compared to those resulting from the shortest path routing computed by OSPF optimization [14] under high fluctuations of traffic demands.

The rest of this paper is structured as follows. Section II reviews related work. We introduce the joker-capable routing

in Section III and then propose JNHOR as our main routing objective. Section IV describes in detail the algorithm to create joker-capable permutation routings that realize JNHORs. Section V shows simulation results assessing the multipath capability, load distribution and computational complexity of JNHORs. We conclude the paper in Section VI.

II. Related Work

Recent IPFRR solutions designed for intradomain networks with link state routing protocols promise 100% single link fault, single node fault or dual link fault coverage. However, some cause forwarding loops when there are multiple uncorrelated failures in the networks, e.g FIFR [4], [5], ESCAP [7] and ESCAP-DL [9], while others have packets carry overheads associating with faulty network components such as MRC [3], 2DLMRC [8], [6], and LOLS [10].

FIFR [4] first introduces interface-specific forwarding concept, in which packets are routed based on both their incoming interfaces and their destination addresses. FIFR offers 100% single link fault coverage while another version [5] provides full coverage of node faults. However, maintaining two routing tables for each interface in FIFR, the forwarding table and the backwarding table (for the rerouting purpose), risks forwarding loops under multiple independent failures as described in [15]. Likewise, ESCAP [7] and ESCAP-DL [9] share with FIFR the concept of interface-specific forwarding except that their routing options are not necessarily bound to the shortest paths. ESCAPE and ESCAPE-DL offer 100% coverage for single link/node fault and dual link faults, respectively. Unlike FIFR, both proposals do not maintain routing tables per interface. Instead, interfaces of a node are classified as backups or primaries beforehand and the routing table (per node) is modified to process types of incoming interfaces of packets.

MRC [3] uses multiple routing tables that cover all possible single failures. Upon detecting a failure on the connected link, the affected node selects another routing table to avoid the network interruption and marks the routing table index in its packets. Other nodes will examine such index in incoming packet and will select the same configuration. The other version of MRC, 2DLMRC [8], guarantees full coverage for dual link faults. Like MRC, IDAG [6] can be resilient under all single link faults by building two node/link-independent configurations on a given topology, called the red tree and the blue tree. This scheme, by allowing nodes to detect failure through the incoming interface of packets, needs only 1 bit for packet marking. IDAG also avoids forwarding loops under arbitrary failures since packets may be transferred from one tree to another at most once. Different from MRCs, LOLS [10] uses a few bits in the IP header to store a minimum set of failed links, called *blacklist*, which a packet encounters on its forwarding path. When receiving a packet with the blacklist, the node will consider the list to find a loop-free path to forward that packet.

Since the modifications required by these methods will typically be costly, adoption in traditional IP networks is limited. This motivates us to design our permutation routing with joker links that strictly complies with the current IP forwarding paradigm, while offering high link fault coverage. Furthermore, our method shares with LFA the important design property, in which shortest path routing is maintained so that the method can work with the standard link state routing protocols, but gives much higher failure coverage by using permutation routing as the loop-free criterion. This work is distinguished from the previous Permutation Routing study [11] in that it applies to IP networks where routers are featured with the functionality that avoids U-turn on their interfaces, meaning that incoming packets would never be sent back on the same interface it was received. Like our scheme, Protection Routing (PR) [16] also employs joker links to increase routing robustness. Our method, however, is designed for the distributed routing systems due to its low complexity while PR is only suitable for small scale networks with a centralized routing systems.

III. Joker-capable Next-Hop Optimality Routing

This section will first explain how we can improve robustness for traditional IP routings through employing joker links. In order to keep our description comprehensible, we introduce networking terms that we will use in the rest of this paper. A *routing* is an assignment of a set of next-hops for each traffic source towards a certain destination node. In a routing, node j is called next-hop of node i if there exists a directed link between node i and node j, denoted by $(i \rightarrow j)$. A node may employ some of its next-hops as primary next-hops (connected by primary links) that are used for packet forwarding in normal operation (fault-free case). The rest are set as backup next-hops (connected by backup links) and only used in case all primary next-hops of that node are unavailable. For a destination, a node with *at least* two next-hops (either primary or backup) is called protected node.

A. Joker-capable Routings

An ideal robust routing against single failures requires that all S-D pairs should have *at least* two next-hops. However, the traditional IP routing can hardly provide the ideal condition; last-hop problem [12] is the famous example for such restriction. Fortunately, the full coverage of single link faults can be achieved if we introduce joker links. Simply put, a joker link is a bidirectional forwarding link in a routing. We denote by $(i \leftrightarrow j)$ the joker link between node i and node j. In other words, i is the next-hop of j and j is also the next-hop of i. Technically, next-hops on joker links of a node should be always set as backups. The beneficial property of joker links is further explained in the following example.

Fig. 1 shows a simple network topology, with 4 different routings for destination node 1. The network topology has 6 nodes and 8 bidirectional links labeled with their link weights. Fig. 1b is a shortest path tree (SPT) rooted at node 1, in which no source node is protected (gray nodes). Other routing methods, e.g. LFA, can improve the SPT by growing *non-shortest* path branches to form a better-connected routing graph, mandatorily a directed acyclic graph. Fig. 1c is such an

instance. Although Fig. 1c is obviously the most robust loop-free routing graph that the traditional IP routing can provide on the given topology, two of five source nodes, node 2 and node 4, are unprotected. We observe that node 2 in Fig. 1c can be protected if we let node 2, at the same time, takes node 3 as its next-hop. In other words, we have established a joker link between node 2 and node 3. We might think that node 4 in Fig. 1c could be protected by joker link $(4 \leftrightarrow 5)$. Adding joker link $(4 \leftrightarrow 5)$, however, would break SPT-constraint since SPT links are not expected to be backup links. Therefore, the only way to protected all nodes in the routing towards node 1 is to relax the shortest path compatibility constraint and then set up a joker link between node 4 and node 5 as in Fig. 1e. Two last routings are called joker-capable routings.

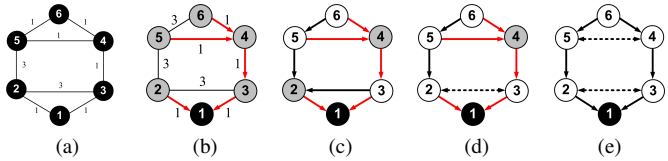


Fig. 1: (a) Network topology. (b) An SPT rooted at node 1. (c) A directed acyclic graph, which contains the SPT, rooted at node 1. (d) A joker-capable routing that contains the SPT. (e) A joker-capable routing that does not contain the SPT.

Clearly, the SPT compatibility constraint will affect resulting robustness of joker-capable routings. We further show that topology structure also restricts fast rerouting capability of joker-capable routings. Namely, the following theorem proves that joker-capable routings fail to cover all possible link faults for a ring topology with its number of nodes greater than 4.

Theorem 1: The maximum single link fault coverage given by joker-capable routings across all destinations on a ring topology with number of nodes $n \geq 3$ is $\frac{2}{(n-1)}$.

Proof: There are n nodes and n links in a ring topology with $n \geq 3$ nodes. A joker-capable routing on that topology towards a certain destination d has $(n-1)$ source nodes, all of which may have one primary next-hop. Hence, there exists one link connecting two nodes i and j $(i, j \neq d)$ that can be set as a joker link. In such joker-capable routing, there will have a maximum of two S-D pairs protected out of $n-1$ pairs. As a result, we obtain a fraction of $\frac{2}{(n-1)}$ S-D pairs protected for all destinations under single link faults. ■

It is obvious that a ring topology with 4 nodes is the smallest topology unit that restricts full single link fault coverage capability given by joker-capable routings. In practice, joker-capable routings also can not provide the ideal fault coverage in more complicated topologies because of the presence of ring structures. Consequently, our goal is to construct joker-capable routings with minimum numbers of unprotected S-D pairs across all destinations. This becomes our main routing objective which will be presented shortly.

B. Joker-capable Next-Hop Optimality Routing (JNHOR)

As mentioned earlier, a joker link is simply a bi-directional forwarding link in a given routing. More generally, we define it as follows:

Definition 1: Given a routing, a joker link is a bi-directional forwarding link that connects two nodes, both of which currently have the same number of primary next-hop(s) for a given destination.

Different links will be joker links for different destinations. In addition, one node may establish different joker links with other nodes in a routing. However, if such node maintains only one of those joker links and other joker links are turned to be normal directional links towards that node, the number of nodes with at least two next-hops would be kept unchanged for that routing. For example, in a routing towards destination d, node i has one primary next-hop and has two joker links with node j and node k, each of which also has one primary next-hop. In other words, node i has three next-hops while j and k both have two next-hops. If we replace joker link $(i \leftrightarrow j)$ with unidirectional link $(j \rightarrow i)$, the three nodes all have two next-hops. Fig. 2 illustrates this. The joker-capable routing towards destination 1 in Fig. 2a has 3 protected nodes and node 4 has two joker links: $(4 \leftrightarrow 2)$ and $(4 \leftrightarrow 3)$. Meanwhile, in the same topology, node 4 in routings in Fig. 2b and Fig. 2c, both of which also have 3 protected nodes, has only one joker link.

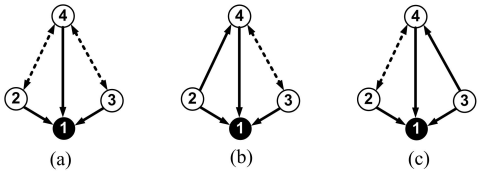


Fig. 2: One node may join in different joker links to other nodes but limiting to only one will not decrease the number of protected nodes

It is obvious that a joker link helps its both ends both have at least two next-hops. Therefore, identifying all possible joker links will increase the robustness for a routing. That leads us to the following optimization criterion for Joker-capable Next-Hop Optimality Routing (JNHOR):

Definition 2: Given a topology, a JNHOR is a joker-capable routing that maximizes the number of S-D pairs that have *at least* two next-hops towards a destination.

A JNHOR in general will not always be compatible with shortest path routing. However, in practice such SPT inclusion constraint helps the method to work with the standard link state routing protocols, e.g. OSPF or IS-IS. In addition, the constraint also is required for traffic engineering purpose, e.g. shortest paths likely provide low transmission latency for voice or video traffic. For that reason, we also define the Shortest Path compatible JNHOR (JNHOR-SP):

Definition 3: Given a topology, a JNHOR-SP is a joker-capable routing that maximizes the number of S-D pairs that have *at least* two next-hops while containing the SPT towards a destination.

Following definitions of the joker-capable routing and its two instances, JNHOR and JNHOR-SP, two questions arise regarding to joker links:

1) How to identify which links to be joker links for maximizing fast rerouting capability?
2) How to avoid single-hop loops on joker links and typical loops under multiple failures in the network?

We will answer those questions in the next section.

IV. Algorithm Design

We design an algorithm that generates joker-capable routings for a given topology. The algorithm includes two tasks: constructing a permutation routing and identifying all joker links for that routing. We first give definition and a generic method to construct permutation routings.

A. Algorithm for Permutation Routing

We model the given topology as graph $G = (V, E)$ where V is the set of nodes, $|V| = N$ and E is the set of undirected links, $|E| = M$. Permutation Routing [11] is based on the observation that any loop-free routing graph can be presented by a sequence of N nodes in a particular order. Specifically, we give the definition of permutation routings as follows:

Definition 4: Given G and the constraint function $\mathbf{C}(u)$, which is defined to realize a selected routing objective, on each node $u \in V$, a permutation routing for destination d is:

1) A sequence of N nodes, called a permutation which is denoted by $\mathbf{P}$, whose positions satisfy constraint function $\mathbf{C}$(u). The destination d is at the left-most position of permutation $\mathbf{P}$.
2) A node can forward its packets to all its neighboring nodes that occur before it in permutation $\mathbf{P}$.

The following example illustrates the definition. We revisit the topology in Fig. 1a and assume that the routing objective is to offer node 5 with two next-hops towards the destination 1, denoted by $\mathbf{C}(5) = 2$. The sequence in Fig. 3a is the permutation routing for this objective, and its corresponding routing graph has been shown in Fig. 1c. Likewise, Fig. 3b is the permutation routing for constraint $\mathbf{C}(5) = 3$. In this case, node 5 is placed after node 6, and therefore can take node 6 as its next-hop. Clearly, the constraint function of a node will determine the position of that node in the permutation routing.

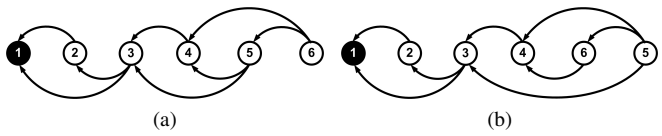


Fig. 3: Illustration for Definition 4

We design a backtracking algorithm to create such permutation $\mathbf{P}$. The key procedure of the algorithm is to assign a node to a variable that represents a position in the permutation so that the assignment satisfies $\mathbf{C}$(u). We show in Fig. 4 such basic assignment procedure. Specifically, we have a set of N variables, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$, in a fixed order from $\mathbf{p}_1$ to $\mathbf{p}_N$. Function `Update` generates domain $\mathbf{D}_i$ for variable $\mathbf{p}_i$. We refer to $\mathbf{D}_i$ as the *candidate set* which consists of nodes that can be assigned to variable $\mathbf{p}_i$. Then another function, called `Select`, will pick one node in $\mathbf{D}_i$ that fulfills $\mathbf{C}$(u) and assign it to variable $\mathbf{p}_i$. In the figure, each pair $\langle \mathbf{p}_i, u_i \rangle$ represents the assignment of the node u_i to variable $\mathbf{p}_i$. The assignment of nodes to a subset of variables $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_i\} \subseteq \mathbf{P}$ given by $\{\langle \mathbf{p}_1, u_1 \rangle, \ldots, \langle \mathbf{p}_i, u_i \rangle\}$ is called *partial* routing permutation with i nodes. For simplicity, we abbreviate it to $\vec{\mathbf{p}}_i$.

This basic assignment procedure has been embedded into **Algorithm** 1 to obtain permutation $\mathbf{P}$. The algorithm calls function `Select` (with built-in constraint function $\mathbf{C}$(u)) which goes through $\mathbf{D}_i$ to find a valid node for the current variable $\mathbf{p}_i$ (line 5). If `Select` succeeds in finding a valid assignment, the algorithm calls function `Update` to generate domain $\mathbf{D}_{i+1}$ (line 10) and proceeds to next variable $\mathbf{p}_{i+1}$. Otherwise, a backtrack step will be executed to revisit the variable $\mathbf{p}_{i-1}$ (line 7). The algorithm terminates if a routing permutation $\mathbf{P}$ of N nodes, also denoted by $\vec{\mathbf{p}}_N$, is found or a failure notification returns if all backtracks are examined but no solution is found under $\mathbf{C}$(u).

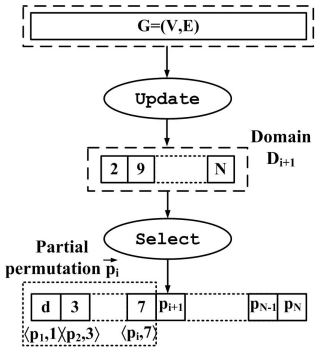


Fig. 4: Basic assignment procedure for variable $\mathbf{p}_{i+1}$

Algorithm 1: `Backtracking`

Input: Topology $G = (V, E)$
Output: Either a solution or failure notification

```
1  i ← 1
2  D_i ← {d}
3  V_i ← V \ {d}
4  while 1 ≤ i ≤ N do
5      p_i ← Select
6      if p_i = null then
7          i ← i − 1
8      else
9          i ← i + 1
10         Update
11 if i = 0 then
12     return failure
13 else
14     return p⃗_N
```

Now we further enrich the routing concept that has been introduced above, by adding joker links. Apparently, the requirement that all forwarding should go in only one direction in the permutation inevitably will leave some nodes with only one routing choice. Then we observe that in carefully

chosen cases, we can let forwarding go in the opposite direction without creating forwarding loops. This observation is the key idea for defining joker-capable routings in terms of permutation routing.

Definition 5: Given a topology with N nodes, a joker-capable permutation routing for destination d is:

1) A permutation denoted by **P**, beginning with destination d, of N nodes, each of which has all its *primary* next-hops occurring before it in permutation **P**.
2) Two nodes connected by a joker link are adjacent nodes in permutation **P**.
3) A node can forward its packets to all its primary next-hops that occur before it in **P** and perform fast rerouting with its adjacent node through its joker link.

The following example shows two permutation routings that include joker links, $2 \leftrightarrow 3$ and $4 \leftrightarrow 5$. Both can be used to represent the joker-capable routing in Fig. 1e. However, only Fig. 5a is valid with respect to Definition 5. The sequence in Fig. 5b is not valid since the joker link connects two nodes, which are not adjacent in the permutation routing.

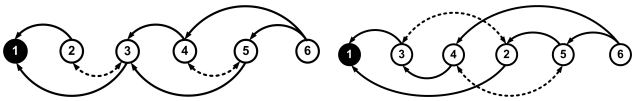


Fig. 5: Valid and invalid joker-capable permutation routings with respect to Definition 5

The forwarding of packets within this structure will be such that packets coming from primary links can use either primary or joker links as their next-hop. Packets coming from joker links are only allowed to enter primary links as their next hop. The above described procedure will guarantee loop-free forwarding also under multiple failures. We will discuss the practicalities of such functionality, and present a proof of freedom from forwarding loops in section IV-E but first proceed with the idea to find all joker links and incorporate it into described framework to produce JNHOR and JNHOR-SP.

B. Joker-capable Next-hop Optimal Routings

1) JNHOR: In JNHOR, computing domains for variables is straightforward. Domain $\mathbf{D}_{i+1}$ for variable $\mathbf{p}_{i+1}$ will consist all nodes that have at least one neighboring node already placed in $\vec{\mathbf{p}}_i$. Otherwise, the resulting routing could be disconnected. Therefore, $\mathbf{D}_{i+1}$ can be written in the recursive form as follow:

$$\mathbf{D}_{i+1} = \mathbf{D}_i \cup \{\ v \in V_i \mid (v,u) \in E\} \setminus \{u\} \qquad (1)$$

where u is the node that has been assigned to variable $\mathbf{p}_i$ in i-th assignment and V_i is a set of nodes from V, excluding all nodes in $\vec{\mathbf{p}}_i$ and $\mathbf{D}_i$.

Given domain $\mathbf{D}_{i+1}$, we will extract a node u from it and assign it to variable $\mathbf{p}_{i+1}$ in a way that the routing resulted from $\vec{\mathbf{p}}_{i+1}$ is the most robustness. In other words, the number of directed edges of resulting routing from $\vec{\mathbf{p}}_{i+1}$ should be maximized:

$$|\mathbf{E}_d(\vec{\mathbf{p}}_{i+1})| = \max_{\forall u \in \mathbf{D}_{i+1}} |\mathbf{E}_d(\vec{\mathbf{p}}_i, \langle \mathbf{p}_{i+1}, u \rangle)| \qquad (2)$$

where $|\mathbf{E}_d(\vec{\mathbf{p}}_{i+1})|$ is the cardinality of $\mathbf{E}_d(\vec{\mathbf{p}}_{i+1})$, which is the set of directed edges formed by $\vec{\mathbf{p}}_{i+1}$. We then derive constraint function **C**(u) to realize expression (2) as follow:

$$\mathbf{C}(u) = \begin{cases} \textbf{True} & \text{if } c[u] = \max_{\forall v \in \mathbf{D}_{i+1}} c[v] \\ \textbf{False} & \text{otherwise} \end{cases}$$

where $c[u]$ denotes the number of outgoing links from u to $\vec{\mathbf{p}}_i$ if u is selected in (i + 1)-th assignment.

There could be more than one node satisfying **C**(u). Let $\mathcal{D}_{i+1}$ be a subdomain of $\mathbf{D}_{i+1}$ containing all such nodes, we have noticed that all nodes in $\mathcal{D}_{i+1}$ have the same number of next-hops, $\max_{\forall v \in \mathbf{D}_{i+1}} c[v]$, placed in $\vec{\mathbf{p}}_i$. If there exists a link connecting any two nodes in $\mathcal{D}_{i+1}$, according to Def. 1, we can set that link as a joker link. We implement joker link identification procedure in function Update of described framework and call it `Update-JNHOR`.

In `Update-JNHOR`, domain $\mathbf{D}_i$ is generated following (1) from line 2 to line 4. We then calculate $\max_{\forall v \in \mathbf{D}_{i+1}} c[v]$ in line 6, upon which we find $\mathcal{D}_i$ from line 8 to line 10. Next we scan through subdomain $\mathcal{D}_i$ to check if there exists a single link connecting any pair of nodes in $\mathcal{D}_i$. If there is the case, we put such pair into $jokerSet$ in line 16 and set variable $isJoker$ true; otherwise false. Because each node should join *at most* one joker link as stated in Section III, *jokerSet* is only allowed to have *at most* two nodes if it is not empty. Furthermore, in order to guarantee *consistent* selection of joker links among nodes in the network, we should choose a pair v_1, v_2 ($(v_1, v_2) \in E$) in $\mathcal{D}_i$ such that the sum of IDs of v_1 and v_2 is maximized.

We then implement function `Select` to pick a node u from $jokerSet$ or from $\mathcal{D}_i$ and place it in the permutation. If $jokerSet$ is not empty, we select one node, say v_1, from that set and assign it to $\mathbf{p}_i$. We then set the next-hop counter of another node in *jokerSet*, say $c[v_2]$ of node v_2, to a large integer, say L, in line 4. This action will guarantee that v_2 will be selected in the next assignment and therefore two nodes connecting by a joker link would be adjacent in the resulting permutation. If *jokerSet* is empty, we pick one node from $\mathcal{D}_i$ and assign it to $\mathbf{p}_i$. To guarantee the consistent choices for all nodes in the network, we always pick one node from *JokerSet* or from $\mathcal{D}_i$ that has the maximum ID.

2) JNHOR-SP: More restrictively, domain $\mathbf{D}_{i+1}$ of an JNHOR-SP includes all nodes that *either* have at least one neighboring node already placed in $\vec{\mathbf{p}}_i$, or do not violate the ordering of nodes in the SPT. Let $c_{sp}[v]$ be the number of shortest path next-hops already appeared in $\vec{\mathbf{p}}_i$ and $n_{sp}[v]$ be the total number of shortest path next-hops that can be calculated from the SPT of node v. The domain $\mathbf{D}_{i+1}$ for variable $\mathbf{p}_{i+1}$ follows the recursive relation:

$$\mathbf{D}_{i+1} = \mathbf{D}_i \cup \{v \in V_i \mid c_{sp}[v] = n_{sp}[v]\} \setminus \{u\} \qquad (3)$$

where u is the node that has been assigned to variable $\mathbf{p}_i$ in i-th assignment and V_i is a set of nodes from V, excluding all nodes in $\vec{\mathbf{p}}_i$ and $\mathbf{D}_i$.

We can reuse constraint function **C**(u) defined for JNHOR to decide assignments in JNHOR-SP because **C**(u) could

Function Update-JNHOR

```
1  D_i ← D_i \ {u}
2  for v ∈ V_i such that (v, u) ∈ E do
3      c[v] ← c[v] + 1
4      D_i ← D_i ∪ {v}
5  V_i ← V_i \ D_i
6  c_max ← max_{∀v∈D_i} c[v]
7  D_i ← ∅
8  for v ∈ D_i do
9      if c[v] = c_max then
10         D_i ← D_i ∪ {v}
11 isJoker ← false
12 jokerSet ← ∅
13 for ∀v1, v2 ∈ D_i do
14     if ∃(v1, v2) ∈ E then
15         isJoker ← true
16         jokerSet ← {v1, v2}
```

Function Select

```
1  if isJoker = true then
2      for v1, v2 ∈ jokerSet do
3          u ← v1
4          c[v2] ← L
5      return u
6      D_i ← ∅
7      jokerSet ← ∅
8  else
9      an arbitrary node u from D_i
10     return u
11     D_i ← ∅
12 return null
```

Function Update-JNHOR-SP

```
1  D_i ← D_i \ {u}
2  for v ∈ V_i such that (v → u) ∈ E_d^spt do
3      c_sp[v] ← c_sp[v] + 1
4  for v ∈ V_i such that (u, v) ∈ E do
5      c[v] ← c[v] + 1
6  for v ∈ V_i do
7      if c_sp[v] = n_sp[v] then
8          D_i ← D_i ∪ {v}
9  V_i ← V_i \ D_i
10 c_max ← max_{∀v∈D_i} c[v]
11 D_i ← ∅
12 for v ∈ D_i do
13     if c[v] = c_max then
14         D_i ← D_i ∪ {v}
15 isJoker ← false
16 jokerSet ← ∅
17 for ∀v1, v2 ∈ D_i do
18     if ∃(v1, v2) ∈ E ∧ (v1 → v2) ∉ E_d^spt ∧ (v2 → v1) ∉ E_d^spt
       then
19         isJoker ← true
20         jokerSet ← {v1, v2}
```

maximize the routing robustness but does not ruin the SPT ordering. Like JNHOR, there could be more than one node in domain $\mathbf{D}_{i+1}$ that satisfies $\mathbf{C}$(u) and we group those nodes in a subdomain, also called $\mathcal{D}_{i+1}$. A joker link in JNHOR-SP, if exists, will be the one connecting two nodes in $\mathcal{D}_{i+1}$ and does not contain any of SPT links.

We create Update function for JNHOR-SP, called `Update-JNHOR-SP`. The candidate set $\mathbf{D}_i$ is calculated following (3), from line 1 to line 8 and the joker link identification procedure from line 10 to line 20. The Select function for JNHOR-SP is identical to that of JNHOR.

C. Computational complexity

The construction of JNHOR and JNHOR-SP are similar to that of typical permutation routings [11], and the added joker link identification procedure works with either empty or non-empty *JokerSet*. Therefore, both of joker-capable permutation routings will also be backtrack-free.

In practical networks, the size of $\mathcal{D}_i$ is typically small ($|\mathcal{D}_i| << N$). The computational complexity for N permutations towards N destinations would be $O(M+N)N$ for a sparse topology and $O(N^3)$ for a dense one.

D. Optimality

We specify earlier that the topology structure may pose limitations on failure protection. However, we show in the following proposition that if the topology allows to construct a routing providing full coverage of single link faults, our algorithm will find the corresponding permutation routing.

Proposition 1: Assume that our algorithm has resulted in a permutation P without backtracking for a given destination d. If there exists a permutation P' that gives 100% coverage for destination d, then P will also give 100% coverage.

Proof: See Appendix. ■

E. Loop-freeness

We prove that our JNHOR (or JNHOR-SP) is free of loops, even when the network suffers from multiple independent failures. First, we are facing the single-hop loop caused by a joker link whose all primary next-hops of both ends are simultaneously unavailable. Fortunately, routers, relying on our standard check, can avoid routing packets back on the same interface it was received. Alternatively, loops on joker links can also be avoided if networks use interface-specific forwarding. To this end, each *line-card* in a router will maintain its own *distinct* forwarding table that maps each network destination to eligible outgoing interfaces. When arriving at a node from primary links, packets will be forwarded to either primary links or joker links. However, packets from joker links are only allowed to take next-hops on primary links. Such restriction helps avoid single-hop loops on joker links. Fig. 6 shows routing tables at interfaces of node 2 and node 3 for the joker-capable routings in Fig. 1d and in Fig. 1e.

We further prove that JNHOR is free of typical loops when packets are forwarded over different joker links due to the presence of multiple independent failures.

Node	Incoming interface	Next-hop	Primary
2	4 → 2	1	Yes
		3	No
	3 → 2	1	Yes

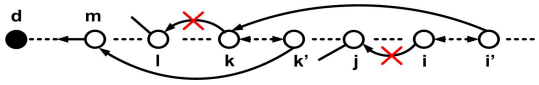
Node	Incoming interface	Next-hop	Primary
3	4 → 3	1	Yes
		2	No
	2 → 3	1	Yes

Fig. 6: Routing tables for interfaces.

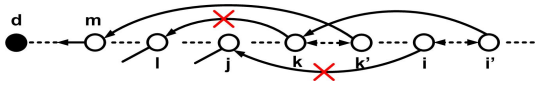
Theorem 2: Given a topology, a JNHOR (or a JNHOR-SP) towards a destination equipped with a single-hop avoidance technique will be free of all loops.

Proof: We give the proof by investigating the operation of extended permutation routing **P** that realizes JNHOR under multiple independent failures of primary next-hops as shown in Fig. 10. In addition, we assume that each node has at least two next-hops (at least one primary next-hop). We write $j < i$ to denote that j occurs before i in **P**.

We consider a certain node i whose only primary next-hop j ($j < i$) has been failed. Due to joker link $(i \leftrightarrow i')$, i would perform fast rerouting through i'. Upon receipt of packets from i, i' sends those packets to its primary next-hop k ($k < j$ as in Fig. 7a or $k > j$ as in Fig. 7b). Without loss of generality, we further assume that the second failure occurs on the only primary next-hop of k. The fast rerouting is repeated at k though joker link $(k \leftrightarrow k')$ and then reaching m, a primary next-hop of k'. We observe that the ordering of nodes in an extended permutation routing will allow packets to be rerouted through its joker link and definitely never re-enter the affected node from any other links. Hence, packets could be normally forwarded (from m) to destination d if there is no other failures or be rerouted as the same described manner but finally reaching d.



(a) $k < j$



(b) $k > j$

Fig. 7: A joker-capable permutation routing with multiple failures

■

V. Performance Analysis

We assess routing robustness of JNHOR and JNHOR-SP in terms of their path diversities and link utilizations under high fluctuations of traffic demands in comparison with their counterparts, ANHOR and ANHOR-SP [11], respectively, and two currently implemented methods, ECMP and LFA.

A. Evaluation Setup

1) Network topologies: We select six representative network topologies from the Rocketfuel project [17] for our evaluations. For each topology, we remove all nodes that will not contribute on routing (e.g. single degree node). The refined topologies are bi-connected graphs, listed in Table I in increasing order of their average node degrees.

TABLE I: Network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	50	194	3.88
3967	Exodus(us)	72	280	3.89
1755	Ebone(eu)	75	298	4.00
3257	Tiscali(eu)	115	564	4.90
6461	Abovenet(us)	129	726	5.60
1239	Sprint(us)	284	1882	6.62

2) Traffic matrices and link weight settings: For each topology, we first create a base traffic matrix (TM) by using the simplified gravity model [18]. We then generate demand fluctuations by drawing each element, μ, of the base TM from Gaussian distribution $N = (\mu, \sigma^2)$ where variance $\sigma^2 = \alpha\mu$. α is known as the *peakedness* of the traffic and is set to be 1 in our simulations. For better understanding the operating range of routing methods, each one is tested with 50 random TMs with Gaussian variations.

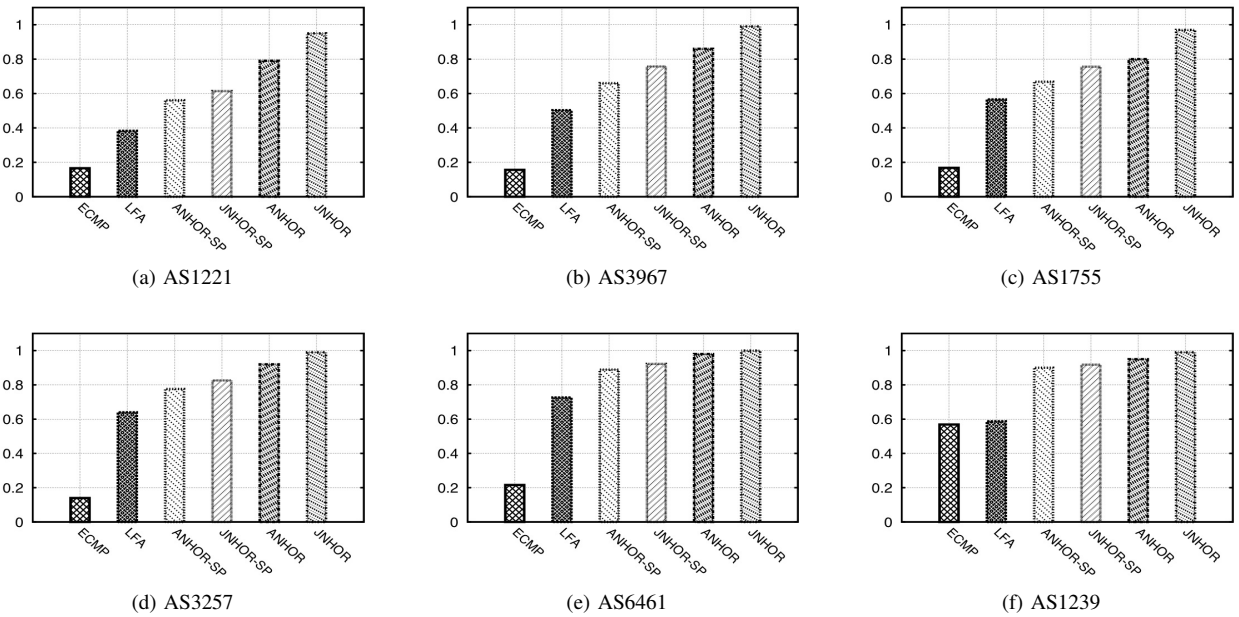
The ECMP and LFA bases their path calculation on link weights. We implement local search heuristic [14] to obtain an optimized link weight setting under the base TM.

3) Traffic load-balancing: In multipath routing, traffic is split among available next-hops. Typically, Equal-Split load-balancing (Equal-Split LB) is well suited for equal-cost multipath but working poorly with unequal-cost multipath [19]. In this paper, we use Distributed Exponentially-weighted Flow Splitting load-balancing (DEFT LB) [13], for unequal-cost multiple paths. Under DEFT LB, split ratios among paths towards a destination at a certain node are proportional to e^{-x_i} where x_i is the difference between the length of path i and the shortest path. Note that Equal-Split LB and DEFT LB result in identical split ratios under equal-cost multiple paths.

4) Flow simulation: Flow arrivals follow a global Poisson process in which the inter-arrival time of two consecutive flows has an exponential distribution with parameter λ calculated from the input traffic matrix. The size of a flow is drawn from a truncated Pareto distribution and flow size distribution are based on packet traces presented in [20]. Sum of all flows on a link is penalized by a link cost, which is a function of link utilization [14].

B. Robustness Evaluation

Given our simulation settings, we select two metrics that reflect routing robustness: path diversity and load distribution under various traffic matrices. Path diversity of a routing method represents the level of fault tolerance and load-balancing of a network. Clearly, in order to increase path diversity, we want to minimize the number of S-D pair with one next-hop. However, it has a cost in terms of path inflation which in turn raises concerns on network congestion if traffic is sent over non-shortest paths. For that reason, we also evaluate load distribution resulted from our routings under traffic variations. Accordingly, for a certain TM we measure



(a) AS1221 (b) AS3967 (c) AS1755

(d) AS3257 (e) AS6461 (f) AS1239

Fig. 8: Fraction of nodes with at least two next-hops

network-wide average cost Φ, which is the total sum of individual link costs. A routing gives a more efficient load distribution if it provides lower average cost Φ while traffic does not take considerably long paths compared to shortest paths.

1) Path Diversity: Fig. 8 shows fractions of protected S-D pairs with six routing methods. We observe that JNHOR provides up to more than 99% of protected nodes in most tested topologies, except for AS1221 with 95%. In addition, JNHOR and JNHOR-SP perform significantly better than ANHOR and ANHOR-SP, respectively. On the contrary, ECMP and LFA give limited protection coverages in most cases. In addition, Fig. 8 also shows that the SPT constraint property in JNHOR-SP substantially reduces the number of joker links. Consequently, in terms of protected S-D paris JNHOR-SP, despite of performing better than ANHOR-SP, is lower than ANHOR. Those results confirm that different structures of SPTs for different destinations and the constraint which joker links are not allowed to contain SPT links contributes to the low improvement of JNHOR-SP.

2) Load distribution: Fig. 9 shows average cost Φ against hop-count path stretch factors[1] across three topologies, AS1221, AS1755 and AS3967, with three different routing methods used to route 50 TMs. We limit the number of available next-hops at each node to $K = 4$ because installing more than a few next-hops will not give much benefit with respect to robustness.

Under DEFT LB, JNHOR-SP is as good as ECMP in the three topologies. Both of the routings successfully deliver all flows with small average link utilization and path stretch factors. Note that hop-count path-stretch can be greater than 1 because computing it uses hop-count shortest paths while ECMP uses realistic link weight settings. JNHOR with DEFT LB, however, increases average network cost compared to ECMP. This is not surprising due to two reasons. First, sending traffic on non-shortest paths increases the total load in the network. Second, JNHOR does not include the shortest paths and therefore results in much longer multiple paths, which make DEFT scheme less efficient (e^{-x_i} is too small). We defer a discussion of better suited load-balancing methods for unequal-cost multipath routing to future work.

[1]defined as the ratio of average path lengths to the corresponding shortest path length for each S-D pair in terms of hop-count

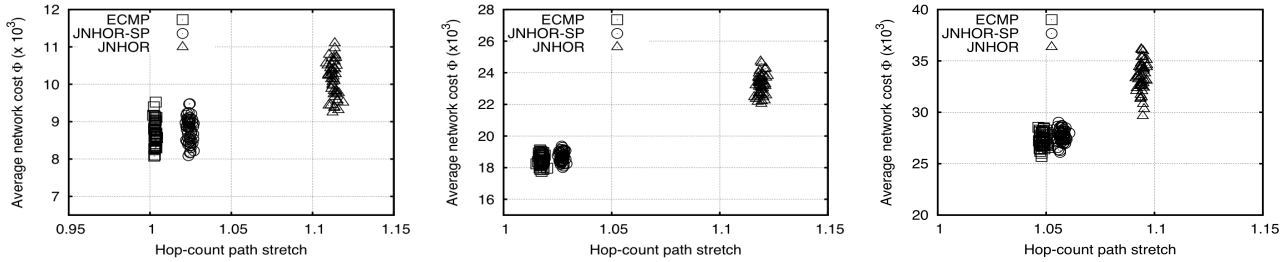
C. Running time

We measure the relative running time of JNHOR and JNHOR-SP to ECMP across six topologies with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. JNHOR has a low running time that is comparable to that of a normal ECMP. For all destinations, the total time difference is less than 10% for all topologies. As for JNHOR-SP, calculating permutations for all destinations takes less than three times of ECMP.

VI. Conclusion

We have presented joker-capable permutation routing as a method to maximize the robustness for IP networks with hop-by-hop forwarding. Our routing methods is proven to be free of loops, does not introduce overheads for IP packets and works with the standard link state routing protocol, e.g. OSPF or IS-IS.

We have evaluated joker-capable permutation routings by simulations on six ISP topologies. The results show that our routings offer fast-rerouting coverage above 95% S-D pairs in tested topologies with typical link weight settings. In addition,



(a) AS1221 (b) AS3967 (c) AS1755

Fig. 9: Average network cost against hop-count path stretch

our routings with DEFT LB provide a good traffic distribution under high traffic demand fluctuations. Measurements of running time show that the methods are computationally feasible.

REFERENCES

[1] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," *IEEE INFOCOM*, vol. 00, pp. 2307–2317, October 2004.

[2] X. Y. M. Gjoka, V. Ram, "Evaluation of IP fast reroute proposals," *IEEE COMSWARE*, vol. 12, Jan. 2007.

[3] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple routing configurations for fast IP network recovery," *IEEE/ACM Transaction on Networking*, vol. 17, pp. 473–486, April 2009.

[4] 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, pp. 359–372, April 2007.

[5] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C.-N. Chuah, "Failure inferencing based fast rerouting for handling transient link and node failures," *IEEE Global Internet*, Mar. 2005.

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

[7] K. Xi and H. J. Chao, "IP fast rerouting for single-link/node failure recovery," *Broadnets*, pp. 142–151, Sept. 2007.

[8] A. F. Hansen, O. Lysne, T. Cicic, and S. Gjessing, "Fast proactive recovery from concurrent failures," *ICC*, pp. 115–122, 2007.

[9] K. Xi and H. J. Chao, "IP fast reroute for double-link failure recovery," *GLOBECOM*, pp. 1–8, 2009.

[10] G. Robertson and S. Nelakuditi, "Handling multiple failures in IP networks through localized on-demand link state routing," *IEEE Transactions on Network and Service Management*, 2012.

[11] H. Q. Vo, O. Lysne, and A. Kvalbein, "Permutation routing for increased robustness in IP networks," *Networking 2012*, vol. 1, pp. 217–231, 2012.

[12] G. Schollmeier, J. Charzinski, and A. Kirstadter, "Improving the resilience in IP networks," *High Performance Switching and Routing 2003 HPSR Workshop*, pp. 91–96, 2003.

[13] D. Xu, M. Chiang, and J. Rexford, "DEFT: Distributed exponentially-weighted flow splitting," *IEEE INFOCOM*, pp. 71–79, May 2007.

[14] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," *IEEE INFOCOM*, pp. 519–528, 2000.

[15] G. Enyedi and G. Rétvári, "A loop-free interface-based fast reroute technique," *Next Generation Internet Networks*, pp. 39–44, April 2008.

[16] K.-W. Kwong, R. G. L. Gao, and Z.-L. Zhang, "On the feasibility and efficacy of protection routing in IP networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, Oct. 2011.

[17] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *SIGCOMM Comput. Commun. Rev.*, 2002.

[18] A. Nccui, N. T. S. Bhattacharyya, and C. Diot, "IGP link weight assignment for operational tier-1 backbones," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 789–802, August 2007.

[19] A. Kvalbein, C. Dovrolis, and C. Muthu, "Multipath load-adaptive routing: putting the emphasis on robustness and simplicity," *ICNP*, 2009.

[20] R. Prasad and C. Dovrolis, "Beyond the model of persistent TCP flows: Open-loop vs closed-loop arrivals of non-persistent flows," *Annual Simulation Symposium*, Oct. 2008.

APPENDIX

The Appendix is dedicated to the proof for Proposition 1.

Assume that P does not give 100% coverage. Then at one point, our algorithm will insert a node a into a position in the permutation, such a does not have two routing alternatives. We shall show that in that case the algorithm would have chosen another node, a_n, instead of a, giving a contradiction.

Consider the position that a has in P'. Clearly a has at least two routing alternatives here. Since a has only one routing alternative in P there must be some nodes that are placed before a in P' and that had not been placed when a was in placed P.

Now identify a_n to be the earliest node placed before a in P' and that is placed behind a in P. If a_n has a joker link in P', we call it a'_n. Now if a'_n does not exist, all of the next hops of a_n in P' have been placed before a in P. But then a_n has at least two next hops already placed in P thus should have gotten the position that a got because the algorithm will always select a node with 2 next hops before a node with 1 next hop. If a'_n exists, then it is either placed before a in P, in which case a_n has two next hops already, or it should have been placed together with a_n with a joker link between them.

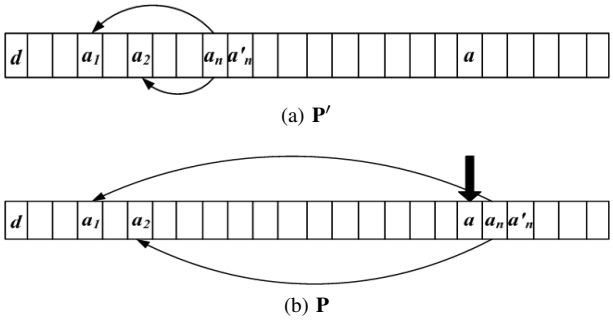


Fig. 10: Illustration of the proof of the Proposition