Increased Robustness with Interface Based Permutation Routing

Hung Quoc Vo, Olav Lysne, and Amund Kvalbein

Simula Research Laboratory
{vqhung,olav.lysne,amundk}@simula.no

Abstract. A prime objective of fault tolerant routing methods is the availability of multiple routing options at each hop. The methods that are currently implemented in IP networks, such as Equal-Cost Multi-Path (ECMP) and Loop Free Alternates (LFA), share the following four properties: First, they work with a hop-by-hop forwarding strategy optimized for the fault free case. Second, they do not require information associated with network faults included in the packet header. Third, they do not form forwarding loops, even under multiple failures in the network. Finally, they are compatible with standard link state routing protocols. However, ECMP and LFA give very poor fault coverage; in most cases fewer than 50% of primary next-hops are protected when using typical link weight settings. This paper presents a new routing method that combines the concept of permutation routing with interface-specific forwarding. Our method results in a routing strategy that strictly adheres to the four stated design properties. Through experiments we show that we protect more than 97% of the primary next-hops for all tested ISP networks.

Keywords: IP networks; Multipath Routing; Resilience; Fault-tolerant

1 Introduction

IP networks are the preferred transportation medium for time-critical services such as online trading, remote monitor and control systems, telephony and video conferencing. Fast recovery from network failures has therefore become an important requirement when designing networks. Currently implemented solutions to the fast reroute problem in IP networks are Equal-Cost Multi-Path (ECMP) and Loop Free Alternate (LFA) [4], both of which share four common properties that are critical for adoption: First, they do not require the network operator to change the traditional hop-by-hop forwarding strategy that is optimized for the fault free case. Second, they do not require addition of fault-information included in the packet header. Third, they do not suffer from routing loops, even when there are multiple faulty components in the network. Finally, they work with the existing standard link state routing protocols such as OSPF or IS-IS. ECMP and LFA are simple but give limited protection against network failures; in most cases they protect below 50% of primary next-hops when using typical link weight settings [17].

Maximizing failure coverage has become a common routing objective for increased robustness in many recently proposed routing schemes. Several solutions, specifically Tunnels [19], Not-via [5], MRC [6] and IDAGs [26], promise full coverage for single network component faults with the cost of altering the traditional IP forwarding. More related to our work, Failure Insensitive Routing (FIR) [3] introduces the interface-specific forwarding concept, which has potential to be deployed with low complexity on modern high-performance routers. FIR also offers full fault coverage for single link faults but it may cause routing loops when there exist multiple failures in the network [20].

In this paper, we propose interface based permutation routing, which combines the concept of permutation routing [21] with interface-specific forwarding [3]. The method aims to *maximize* protection coverage for IP networks by giving multiple loop-free next-hops towards a destination for each incoming interface of a router. Importantly, our routing method shares with ECMP and LFA the four listed properties.

Permutation routing [21] is a new and 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 routing strategy can be expressed as a *permutation* (sequence) of nodes that are involved in traffic forwarding to a destination. The routing is loop-free if packets are forwarded in one direction towards the destination regarding to node ordering in the permutation. Permutation routing only takes the topology information that is collected by link state routing protocols as the input for its construction, and hence no new control plane signaling is needed.

The main focus of this paper is to construct permutation routing of *interfaces* that approximately maximizes the number of primary interface-destination pairs that have more than one available next-hop. Importantly, our routing method can easily be integrated with existing link state routing protocols, OSPF or IS-IS, and can be used to augment shortest path routing tables with additional forwarding entries. We show that our interface based permutation routing can protect more than 97% of the primary next-hops in all tested ISP topologies. This is significantly better than LFA and ANHOR-SP [21], which protect from 21% to 67% and from 29% to 81%, respectively, in the topologies we study. Note that ANHOR-SP is a routing strategy constructed by using the concept of permutation routing, which seeks to maximize protection coverage for traditional IP networks. In this paper we only focus on finding a set of loop-free paths, and leave the important topic of load-balancing across these paths for future work.

The rest of the paper is structured as follows. Section II explains why interface-specific forwarding is beneficial over traditional IP forwarding and proposes iNHOR as our main routing objective. Section III describes two transformation rules to pre-process the topology, resulting a new directed graph that is readily for interface based routing construction. Section IV presents Permutation Routing concept, proposes generic algorithm for permutation construction, and discusses computational complexity management. Section V describes in details the described framework in section IV to create routings that approximate iN-

HOR. Section VI shows simulation results evaluating the path diversity and computational complexity of iNHOR. Section VII reviews related work. Section VIII concludes the paper.

2 Interface Next-Hop Optimal Routing

This section first reviews the interface specific forwarding concept and explains its ability of increasing routing robustness for IP networks. We then introduces Interface Next-hop Optimal Routing (iNHOR) as our main objective function for a robust interface based routing.

To help ease our description, let us introduce networking terms that we will use in the rest of this paper. We refer a directed link from node i to node j, denoted by $(i \rightarrow j)$, to an incoming interface to node j and an outgoing interface of node i. A *routing* is an assignment of a set of next-hops for each destination node for each traffic source. We require that a routing must be loop-free, and hence a given routing corresponds to a Directed Acyclic Graph (DAG) rooted at each destination node consisting of the links and nodes involved in packet forwarding. With multipath routing, each node may have several outgoing links in the DAG for a destination.

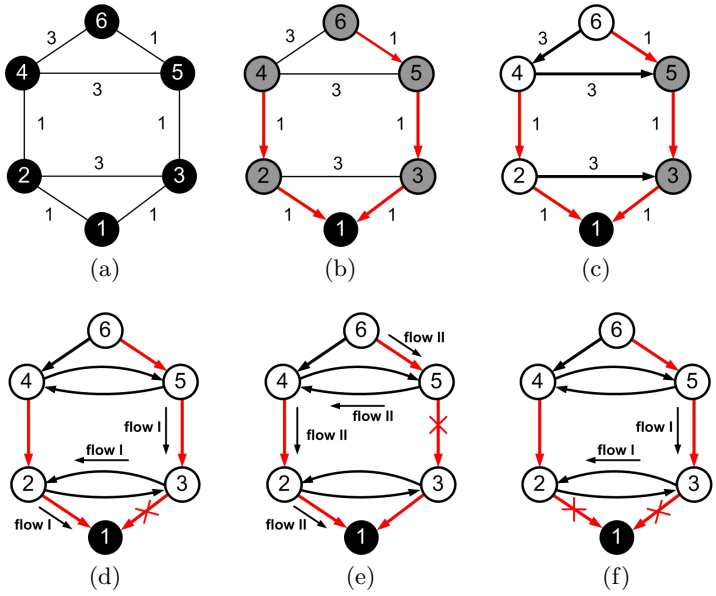


Fig. 1: (a) A topology. (b) An SPT rooted at node 1. (c) A DAG, which contains the SPT, rooted at node 1. Fast rerouting under interface-specific forwarding if (d) link $(1, 3)$ fails, (e) link $(3, 5)$ fails, (f) multiple links fail.

Fig. 1a illustrates a simple topology with six nodes and eight (bidirectional) links with their corresponding link weights. Fig. 1b is a shortest path tree (SPT) rooted at node 1. We say that a node is *protected* if it has *at least* two next-hops, no node in Fig. 1b is protected (gray shadowed nodes). Other routing methods, e.g. ANHOR-SP [21], can improve the protection coverage by adding *non-shortest* path branches to form a better-connected routing graph, mandatorily a directed acyclic graph (DAG). Fig. 1c is such an instance. Obviously, Fig. 1c is the most robust loop-free routing graph rooted at node 1 that the traditional IP routing can provide because adding any more directed link will cause loops. Consequently, we fail in protecting node 3 and node 5 under single failures.

However, node 3 and node 5 of the topology can be protected if interface-specific forwarding is used. Correspondingly, node 3 (in Fig. 1d) will reroute flow I to node 2 under failure of link $(1, 3)$. Upon receipt of rerouting packets, node 2 will forward them to node 1 without causing loops. To do that, interfaces of node 2 and node 3 should be installed with forwarding tables as shown in Fig. 2. Likewise, node 5 (in Fig. 1e) will safely reroute flow II to node 4 under failure of link $(3, 5)$ if interfaces of node 4 and node 5 are also equipped with loop-free forwarding tables. In those tables, next-hops marked with *primary* are used if available. Other next-hops are called secondaries and only used when all primaries are not in service. If there does not exist any next-hop (either primary or secondary) for a specific incoming interface, packets will be *discarded* at that interface as shown in Fig. 1f. We will show how to construct such a forwarding table for each interface in Section III, but we first introduce our main routing objective.

Node	Incoming interface	Next-hop	Primary
2	$4 \rightarrow 2$	1	Yes
		3	No
	$3 \rightarrow 2$	1	Yes

Node	Incoming interface	Next-hop	Primary
3	$5 \rightarrow 3$	1	Yes
		2	No
	$2 \rightarrow 3$	1	Yes
		2	No

Fig. 2: Forwarding tables of incoming interfaces of node 2 and node 3.

2.1 Interface Next-Hop Optimal Routing

For maximizing fault tolerance and load-balancing capabilities of a network with interface-specific forwarding, it is important that the routing offers more than one available next-hop for as many primary incoming interface-destination pairs

as possible. This leads us to the following optimization criterion for Interface Next-hop Optimal Routing (iNHOR):

Definition 1. *Given a routing for a destination, an incoming interface on the SPT is called a primary interface (pI) and an incoming interface* not *on the SPT a secondary interface. An iNHOR is an interface based routing that contains the SPT and maximizes the number of primary interface-destination (pI-D) pairs with* at least *two next-hops.*

By maximizing fault tolerance capability for primary interfaces, iNHOR has a great potential to increase robustness for IP networks under failures. In addition, the SPT inclusion constraint of iNHOR is usually required for traffic engineering purpose, e.g. shortest paths likely provide low transmission latency for voice or video traffic. In the next section, we introduce a new network model that helps construct a loop-free forwarding table for each interface using permutation routing.

3 Network Model

We model the network topology as directed graph $G = (V(G), E(G))$ where $V(G)$ is the set of nodes and $E(G)$ is the set of directed links[1]. Let p and q denote the cardinalities of set $V(G)$ and set $E(G)$, respectively. The directed link $(u \to v)$ consists of head node u and tail node v. Let $\mathcal{L}(G) = (V(\mathcal{L}(G), E(\mathcal{L}(G)))$ be the line graph of G by employing two following rules:

Definition 2. *Each node in $V(\mathcal{L}(G))$ represents a directed link of G*

We denote $[u, v]$ by the node in $V(\mathcal{L}(G))$, which is directed link $(u \to v)$ in $E(G)$. Let $H([u, v])$ and $T([u, v])$ be two operators which apply on node $[u, v]$ to extract the head node and the tail node of corresponding directed link $(u \to v)$.

Definition 3. *Two nodes i, j[2] $\in V(\mathcal{L}(G))$ form directed link $(i \to j)$ in $E(\mathcal{L}(G))$ if and only if $H(i) = T(j)$.*

Following Def. 2 and Def. 3 in that sequence, line graph $\mathcal{L}(G)$ has been identified thoroughly. Let $A_{\mathcal{L}(G)} = (a_{ij})$ be the adjacency-matrix that represents for $\mathcal{L}(G)$:

$$a_{ij} = \begin{cases} 1 \ H(i) = T(j) \\ 0 \text{ otherwise} \end{cases}$$

Following theorems describe some consequences of line graphs. Theorem 1 and Theorem 2 prove that line graphs preserve connectivity, subset relationship and acyclic property of corresponding directed graphs while Theorem 3 provides the cardinalities of node sets and link sets of line graphs.

[1] each link in the topology is modeled as two directed links.

[2] For convenience, we sometimes denote nodes in the new directed graphs with single letters

Theorem 1. *Let G_1 and G be two directed graphs:*

1. *if G is connected, then $\mathcal{L}(G)$ is connected.*
2. *if $G_1 \subseteq G$, then $\mathcal{L}(G_1) \subseteq \mathcal{L}(G)$.*

Proof. 1. see the proof in [22].

2. We prove that $V(\mathcal{L}(G_1)) \subseteq V(\mathcal{L}(G)$ and $E(\mathcal{L}(G_1)) \subseteq E(\mathcal{L}(G))$ both hold. It is trivial that $V(\mathcal{L}(G_1)) \subseteq V(\mathcal{L}(G))$ due to Def. 2. Assume that arbitrary directed links $(a \to b) \in E(G_1)$ and $(b \to c) \in E(G_1)$, then we will have $([a,b] \to [b,c]) \in E(\mathcal{L}(G_1))$. Because $E(G_1) \subseteq E(G)$, then we have $(a \to b) \in E(G)$ and $(b \to c) \in E(G)$. That means $([a,b] \to [b,c]) \in E(\mathcal{L}(G))$. So $E(\mathcal{L}(G_1)) \subseteq E(\mathcal{L}(G))$ holds.

Theorem 2. *The line graph of a directed acyclic graph is also a directed acyclic graph.*

Proof. The proof is by contradiction. We denote D by the given directed acyclic graph and $\mathcal{L}(D)$ by its corresponding directed line graph. We assume that $\mathcal{L}(D)$ contains a loop. Namely, the loop is $[a,b] \to [b,c] \to \ldots \to [y,z] \to [z,a]$ where $a, b, c \ldots y, z$ are nodes of graph D. In addition, the loop implies that there exists a path $a \to b \to c \ldots \to y \to z \to a$ which is another loop of D. Since D is directed acyclic graph, our assumption is broken.

Theorem 3. *Given directed graph from topology G[3], line graph $\mathcal{L}(G)$ is satisfied:*

1. *$|V(\mathcal{L}(G)| = q$.*
2. *$|E(\mathcal{L}(G)| = \sum_{u=1}^{p} d_u^2$ where d_u is the out-degree of node $u \in V(G)$.*

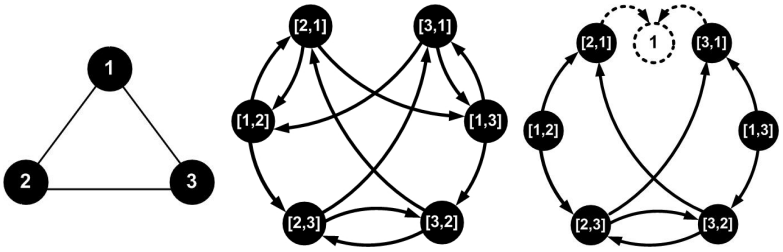
Proof. (1) is trivial due to the Def. 2. We prove (2) by examining each node $u \in V(G)$. Let d_u be the out-degree of node u and $N_G(u) = \{v \in V(G) : (u \to v) \in E(G)\}$ be the set of neighboring nodes of u. We observe that there are d_u directed link incident at u and d_u directed links departing from u. Via Def. 3, each node $[v_k, u] \in V(\mathcal{L}(G))$, where $u \in N_G(v_k)$, will have d_u neighboring nodes. Therefore, in $\mathcal{L}(G)$, there are d_u nodes and d_u^2 directed links involving node u. That means $|E(\mathcal{L}(G))| = \sum_{u=1}^{p} d_u^2$.

We might think that $\mathcal{L}(G)$ is a proper graph for calculating a loop-free routing tables towards a certain destination node. It is, however, infeasible since $\mathcal{L}(G)$ does not include any node that can correspond to the real destination node. For the routing computation purpose, we should modify $\mathcal{L}(G)$. We denote $\mathcal{L}^*(G)$ by the modified version of $\mathcal{L}(G)$. $V(\mathcal{L}^*(G))$ is resulted from adding virtual node u into $V(\mathcal{L}(G))$ and $E(\mathcal{L}^*(G))$ is resulted from augmenting $E(\mathcal{L}(G))$ with virtual directed links that depart from nodes $[v_i, u]$ $(u \in N_G(v_i))$ and terminate at

[3] if there exists directed link $u \to v$, then there also exists directed link $v \to u$.

virtual node u while taking away from $E(\mathcal{L}(G))$ those directed links that depart from nodes $[v_i, u]$ and terminate at nodes $[u, v_j]$ ($v_j \in N_G(u)$). Those removed directed links will not contribute to routing towards u.

$$\begin{cases} V(\mathcal{L}^*(G) = V(\mathcal{L}(G)) \cup \{u\} \text{ where } u \in V(G) \\ E(\mathcal{L}^*(G) = \{E(\mathcal{L}(G) \cup \{([v_i, u] \rightarrow u)\}\} \\ \qquad\qquad\qquad \setminus \{([v_i, u] \rightarrow [u, v_j])\} \end{cases}$$



(a) Topology G (b) Line graph $\mathcal{L}(G)$ (c) Modified line graph $\mathcal{L}^*(G)$

Fig. 3: Topology, its corresponding line graph and its modified line graph towards node 1.

Fig. 3b shows the line graph generated from the topology in Fig. 3a. The line graph includes 6 nodes and 12 directed links. It is obviously larger than the original topology either in the number of nodes or in the number of directed links. Fig. 3c is the modified line graph towards node 1 with the added virtual node 1 and two virtual links in dotted lines.

We can apply an existing node based routing algorithm for traditional IP forwarding on the directed graph $\mathcal{L}^*(G)$ to extract routing entries at each node in $V(\mathcal{L}^*(G))$ towards destination u. The resulting routing table at each node is actual routing table for each incoming interface. In the next section, we propose a new routing method to construct robust interface based routings for IP networks.

4 Permutation Routing

Given modified line graph $\mathcal{L}^*(G)$ towards destination d, we will seek for a routing graph for that destination, denoted by $\mathcal{R}_d = (V, E_d)$ where $V = V(\mathcal{L}^*(G))$ and $E_d \subset E(\mathcal{L}^*(G))$. In $\mathcal{R}_d$, node j is called a *next-hop* of node i if there exists a directed link between node i and node j. It is required that $\mathcal{R}_d$ must be a DAG rooted at destination d. Without loss of generality, we look at the assignment of next-hops for each destination individually.

4.1 Permutation Routing

Routing function $\mathcal{R}_d$ contains all paths to d, each of which is a sequence of nodes from a specific source node to d. At each node, packets are forwarded to a next-hop in such a sequence. In the rest of this section, we describe how a permutation routing can be used as a tool to find such sequences with the goal of realizing iNHOR.

Definition 4. *For given line graph $\mathcal{L}^*(G)$, permutation P of nodes is an arrangement of all nodes in $V(\mathcal{L}^*(G)$ into a particular order.*

We write $j < i$ to denote that j occurs before i in P. Our goal is to construct permutations that realize a certain routing strategy.

Definition 5. *Permutation P is a* permutation routing *for $\mathcal{R}_d$ if all next-hops of each node occur before it in P: $\forall(i \to j) \in E_d : j < i$.*

According to this definition, destination node d will always be the first node in a routing permutation for $\mathcal{R}_d$. Nodes further out in the permutation routing will be topologically farther from the destination. Following lemma confirms the existence of such sequences of *all* nodes in $\mathcal{R}_d$.

Lemma 1. *Any loop-free routing function $\mathcal{R}_d$ can* always *be represented by a permutation routing in which d is at the left-most position.*

Proof. A loop-free routing function $\mathcal{R}_d = (V, E_d)$ is a DAG, rooted at d. Let arrange $i \in V$ into a sequence such that if E_d contains a directed link $(i \to j)$, then j appears before i in that sequence. Such an arrangement can be calculated by a topological sort algorithm [13].

Destination d is the only node that does not have any outgoing link. Following the above ordering, node d, hence, has been placed at the left-most position of the sequence.

In general, there could be more than one valid permutation routing for one routing graph $\mathcal{R}_d$. Starting with a permutation routing P, another permutation routing P' can be generated by swapping *two consecutive* nodes that are not connected by a directed link to each other.

In the reverse process, routing tables can be generated from a permutation routing, given a *forwarding rule* that defines the relationship between neighboring nodes. For now, we consider a *greedy forwarding rule* for constructing the routing table, in which all out-neighbors of a node that occur before it in the permutation routing are installed as next-hops. This will maximize the potential for load balancing and failure recovery. More restrictive forwarding rules could also be considered, which would result in a sparser DAG. This can sometimes be beneficial in order to avoid excessively long paths, or to limit the number of next-hops for a particular destination.

Since permutation routing has been proven to represent a routing function, such routing function will be identified if we can construct its corresponding permutation routing. Our goal is to find a permutation routing that can realize iNHOR. This problem is believed to be NP-hard. In the next section, we present an algorithm that produces permutation routings that approximate iNHOR.

4.2 Generic Algorithms

Let $P = \{p_1, p_2, \ldots, p_N\}$ be a set of N variables in a *fixed order* from p_1 to p_N, with respective domains $D = \{D_1, D_2, \ldots, D_N\}$ where N is the number of nodes in $\mathcal{L}^*(G)$. We refer to D_i as the *candidate set* for each variable p_i. A candidate set consists of the nodes that can be assigned to variable p_i.

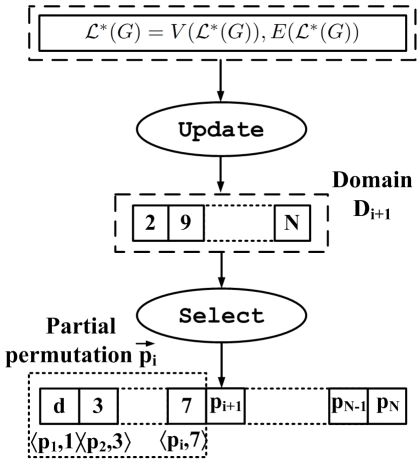


Fig. 4: Basic assignment procedure for variable p_{i+1}

Permutation routing P is constructed by successively assigning a node $u \in D_i$ to each variable p_i. Such assignment is said to be valid if it satisfies a specific *constraint function* $C(\mathrm{u})$ which is defined to realize the selected routing objective. Fig. 4 illustrates the basic assignment procedure for variable p_{i+1} in which two key functions `Update` and `Select` work as filters to control the assignment. In the figure, each pair $\langle p_i, u_i\rangle$ represents an assignment of the node u_i to variable p_i. The assignment of nodes to a subset of variables $\{p_1, p_2, \ldots, p_i\} \subseteq P$ given by $\{\langle p_1, u_1\rangle, \ldots, \langle p_i, u_i\rangle\}$ is called *partial* routing permutation with i nodes. For simplicity, we abbreviate it to $\vec{p}_i$.

This basic assignment procedure has been embedded into the well-known backtracking algorithm (**Algorithm** 1) to obtain permutation routing P. The algorithm calls function `Select` (with built-in constraint function $C(\mathrm{u})$) which goes through D_i to find a valid node for the current variable p_i (line 4). If `Select` succeeds in finding a valid assignment, the algorithm calls function `Update` to generate domain D_{i+1} (line 9) and proceeds to next variable p_{i+1}. Otherwise, a backtrack step will be executed to revisit the variable p_{i-1} (line 6). The algorithm terminates if a routing permutation P of N nodes, also denoted by $\vec{p}_N$, is found or a failure notification returns if all backtracks are examined but no solution is found under $C(\mathrm{u})$.

If constraint function $C(\mathrm{u})$ allows it, the backtracking algorithm will find *one* permutation routing P among all possible solutions by searching through

the *search space* shaped by the number of variables in P and their domains of values. It is known that the complexity of backtracking algorithm varies in a wide range. In the best case, when we do not need any backtrack step (backtrack-free) is $O(t \times N)$ where t is the total computational complexity of functions `Update` and `Select`. In another extreme, if there is only one solution and we always make the "wrong" choice, the complexity would be $O(t \times N!)$. Hence, it is important guide the search to avoid exploring the entire search space. In the implementation described in the next section, we use two main mechanisms to reduce the search space:

1. $C(\mathrm{u})$ should be simple to reduce the computational complexity.
2. Domain D_i can be limited by taking $C(\mathrm{u})$ into account.

Algorithm 1: `Backtracking`

Input: Line graph $\mathcal{L}^*(G)$
Output: Either a solution or failure notification

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

5 Approximate iNHOR

In this section, we propose a heuristic for generating permutation routings that can approximate iNHOR. We call it AiNHOR. To this end, we first present how to construct constraint function $C(\mathrm{u})$ from which a corresponding permutation routing is constructed by using the described algorithm framework.

5.1 Constraint function $C(\mathrm{u})$

We denote $R_d^{spt} = (V, E_d^{spt})$ by the SPT towards destination d which is constructed on topology G. To construct permutation routing for AiNHOR, we need the modified line graphs of both G and R_d^{spt}, denoted by $\mathcal{L}^*(G)$ and $\mathcal{L}^*(R_d^{spt})$,

respectively. According to Theorem 1 and Theorem 2, $\mathcal{L}^*(R_d^{spt})$ is connected, $\mathcal{L}^*(R_d^{spt}) \subseteq \mathcal{L}^*(G)$ and $\mathcal{L}^*(R_d^{spt})$ is a DAG towards d.

To achieve a robust interface based routing while being compatible with the SPT, the resulting routing from its corresponding permutation must contain $\mathcal{L}^*(R_d^{spt})$. From the Def. 5 partial routing permutation $\vec{p}_i$ represents a loop-free routing sub-graph towards destination d, denoted by $\mathcal{R}_d^i = (V(\vec{p}_i), E(\vec{p}_i))$ where $V(\vec{p}_i)$ is the set of i nodes in $\vec{p}_i$ and $E(\vec{p}_i)$ is the set of directed edges formed by applying the greedy forwarding rule defined in Section 4 on $\vec{p}_i$. Then the node selected for variable p_{i+1} to form partial permutation routing $\vec{p}_{i+1}$ should be the node with the maximum number of out-neighbors already placed in $\vec{p}_i$. In addition, $\vec{p}_i$ must contain $R_d^{spt,i+1}$, which is a subset of $\mathcal{L}^*(R_d^{spt})$ after the $(i+1)$-th assignment. Correspondingly, the number of directed edges of the routing sub-graph formed by partial routing permutation $\vec{p}_{i+1}$, from assignment $\langle p_{i+1}, u\rangle$, must be maximized:

$$|E(\vec{p}_{i+1})| = \max_{\forall u \in D_{i+1}} |E(\vec{p}_i, \langle p_{i+1}, u\rangle)| \tag{1}$$

and

$$R_d^{spt,i+1} \subseteq \mathcal{R}_d^i \tag{2}$$

For a more efficient implementation, we maintain a counter, say $c[u]$, for each node u. This counter denotes the number of outgoing links from u to $\vec{p}_i$. In other words, $c[u]$ corresponds to the number of next-hops node u will have if it is selected as the next assignment in the routing permutation. We derive constraint function C(u) to realize expression (1) and (2) as follow:

$$C(u) = \begin{cases} \textbf{True} & \text{if } c[u] = \max_{\forall v \in D_{i+1}} c[v] \\ & \text{and } \mathcal{R}_d^{spt,i+1} \subseteq \mathcal{R}_d^i \\ \textbf{False} & \text{otherwise} \end{cases}$$

Apparently, constraint function C(u) is complicated and thus may not result in an efficient search. We will make C(u) more simple by not using expression (2) in C(u). Instead, expression (2) will be used to reduce domain D_{i+1} in expression (1). Let D_{i+1}^* be the reduced domain readily for assignment $(i+1)$-th, the new constraint function C^*(u) is re-written as follow:

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

Because D_{i+1}^* is usually smaller than D_{i+1}, the search space of our problem would be reduced. In the next section, we describe in details how we find domain D_i^*, from which we select a node satisfying C^*(u).

5.2 Search space reduction

In a connected graph, expression (1) implies that the domain D_{i+1} includes all nodes that have *at least* one topological neighbor (following the directed edge) in $\vec{p}_i$. The domain is, therefore, updated following the recursive relation:

$$D_{i+1} = D_i \cup \{\ v \in \mathcal{V}_i \mid (v \to u) \in E(\mathcal{L}^*(G))\} \setminus \{u\} \tag{3}$$

where u is the node that has been assigned to variable p_i in the i-th assignment and $\mathcal{V}_i$ is a set of nodes from $V(\mathcal{L}^*(G))$, exluding all nodes in p_i and D_i.

Domain D_{i+1} is usually large when i increases. We will divide D_{i+1} into smaller domains, on which our search would be more efficient. We have noticed that D_{i+1} may contain *three* types of nodes. First, they are nodes in $V(\mathcal{L}^*(R_d^{spt}))$ which have *all* their next-hops in $V(\mathcal{L}^*(R_d^{spt}))$ already placed in $\vec{p_i}$, denoted by D_{i+1}^a. Second, they are nodes which are *not* in $V(\mathcal{L}^*(R_d^{spt}))$, denoted by D_{i+1}^b. Third, they are nodes in $V(\mathcal{L}^*(R_d^{spt}))$ which *do not* have *all* their next-hops in $V(\mathcal{L}^*(R_d^{spt}))$ already placed in $\vec{p_i}$, denoted by D_{i+1}^c. Note that those nodes in $V(\mathcal{L}^*(R_d^{spt}))$ correspond to directed links of R_d^{spt} and those nodes *not* in $V(\mathcal{L}^*(R_d^{spt}))$ are directed links of the topology that do not belong to R_d^{spt}.

For a node v, let $c_{sp}[v]$ be the number of next-hops in $V(\mathcal{L}^*(R_d^{spt}))$ placed in $\vec{p_i}$ and $n_{sp}[v]$ be the total number of next-hops in $\mathcal{L}^*(R_d^{spt})$, sub-domain D_{i+1}^a for variable p_{i+1} follows the recursive relation:

$$D_{i+1}^a = D_i^a \cup \{v \in D_{i+1} \mid c_{sp}[v] = n_{sp}[v]\} \tag{4}$$

Sub-domain D_{i+1}^b is calculated as follow:

$$D_{i+1}^b = D_i^b \cup \{v \in D_{i+1} \mid v \notin V(\mathcal{L}^*(R_d^{spt}))\} \tag{5}$$

Function Update

```
1  D_i ← D_i \ {u}
2  for v ∈ V_i such that (v → u) ∈ E(L*(G)) do
3  |  c[v] ← c[v] + 1
4  |  D_i ← D_i ∪ {v}
5  V_i ← V_i \ D_i
6  for v ∈ D_i do
7  |  if (v → u) ∈ E(L*(R_d^spt)) then
8  |  |  c_sp[v] ← c_sp[v] + 1
9  |  |  if c_sp[v] = n_sp[v] then
10 |  |  |  D_i^a ← D_i^a ∪ {v}
11 |  else
12 |  |  D_i^b ← D_i^b ∪ {v}
13 c_max^a ← max_{∀v∈D_i^a} c[v]
14 c_max^b ← max_{∀v∈D_i^b} c[v]
```

We implement the described update domain process in the `Update` function. We first remove node u, which has just been added in the permutation routing, from D_i in line 1. The **for** loop of line 2-4 considers all nodes v in $\mathcal{V}_i$ such that

$(v \to u) \in E(\mathcal{L}^*(G))$ to update D_i following (3) and counter $c[v]$. Another **for** loop of lines 6-12 goes through all nodes in D_i and divides them in two groups: D_i^a following (4) and D_i^b following (5). As the input for constraint function $C^*(\mathrm{u})$, $\max_{\forall v \in D_i^a} c[v]$ and $\max_{\forall v \in D_i^b} c[v]$ are calculated in lines 13-14.

We then select one node from one of two domain groups that satisfies $C^*(\mathrm{u})$ and place it in the permutation. We have two strategies: *first* choosing a node in D_i^a if it is not empty or *first* choosing a node in D_i^b if it is not empty. The resulting routing in both cases will contain $\mathcal{L}^*(R_d^{spt})$. However, the latter will increase the number of next-hops for primary links because it places in the permutation *all* possible secondary links, which can become next-hops for primary links, before primary links. We implement this selection strategy in function `Select`.

The `Select` function will realize constraint function $C^*(\mathrm{u})$ with two steps in a strict order. We will first maximize number of next-hops for primary links by going through sub-domain D_i^b to select a node that satisfies $C^*(\mathrm{u})$ (line 2-6). Note that $C^*(\mathrm{u})$ is simply a comparison between $c[u]$ and pre-computed c_{max}^b (line 5). If D_i^b is empty, we will select a node in D_i^a that satisfies $C^*(\mathrm{u})$ (lines 8-12). Again, $C^*(\mathrm{u})$ is simply a comparison between $c[u]$ and pre-computed c_{max}^a (line 11). Returned node u is assigned to variable p_i in **Algorithm** 1.

Function Select

```
1  if D_i^b ≠ ∅ then
2      while D_i^b ≠ ∅ do
3          an arbitrary node u from D_i^b
4          D_i^b ← D_i^b \ {u}
5          if c[u] = c_max^b then
6              return u
7  else
8      while D_i^a ≠ ∅ do
9          an arbitrary node u from D_i^a
10         D_i^a ← D_i^a \ {u}
11         if c[u] = c_max^a then
12             return u
13 return null
```

5.3 Time complexity

The computational complexity of AiNHOR is the product of the average number of operations to make a move between two successive states and the total number of states visited in the search space.

Proposition 1. *The selection rule defined in function* `Select` *gives a backtrack-free algorithm for all connected input directed graph.*

Proof. The proof is by contradiction. Assume that the algorithm with the selection rule defined in function `Select` is not backtrack-free. This could happen in two following cases:

(i) C^*(u) returns **False** at some iteration for all $u \in D^b_{i+1}$ or for all $u \in D^a_{i+1}$ if D^b_{i+1} is empty. That can not happen because all nodes in domain D^b_{i+1} or domain D^a_{i+1} always have at least one next-hop in $\vec{p_i}$.

(ii) Both D^b_{i+1} and D^a_{i+1} are empty in some iteration. We show here that both D^b_{i+1} and D^a_{i+1} can never be empty at the same time before all nodes have been placed in the permutation. If this is the case, there exists only candidate nodes in $\mathcal{L}^*(R^{spt}_d)$ but do not have all its shortest path descendants in $\vec{p_i}$. In other words, we can follow DAG $\mathcal{L}^*(R^{spt}_d)$ from any node that has not been placed and always find a next-hop node that is not placed in $\vec{p_i}$. But this is impossible: since $\mathcal{L}^*(R^{spt}_d)$ is connected and loop-free, any path from a particular nodes will eventually reach the destination, which is the first node that was placed in the permutation.

The computational complexity of AiNHOR towards one destination includes two parts: the shortest path calculation using Dijkstra's algorithm and the routing permutation construction. Due to the property of backtrack-freeness, with sparse topologies the computational complexity of the second part towards *one* destination would be $O(|E(\mathcal{L}^*_(G))| + q \times |D|)$ where $|D|$ denotes the average size of the domains. In dense topologies, the total computational complexity of calculating routing permutations for *all* destinations can approach $O(q^3)$ where q is the number of directed links in the given topology. The backtrack-free property also gives a low memory consumption because it does not need to store temporary partial permutations.

Such running time can be reduced more by following observation. A node does not necessarily produce a permutation routing of N nodes towards each destination because it only needs to know which are its primary next-hops and its backup next-hop on its joker link (if the joker link exists). Therefore, given destination j each node should create a permutation of $P_j(i)$ (or $P_j(i)+1$) nodes where $P_j(i)$ is the position of node i in the permutation P towards destination j. The average length of the permutations calculated by node i for all destinations would be $\overline{N_i} = \frac{1}{N}\sum_{j=1}^{N} P_j(i)$ and the average length of the permutations calculated by N sources for all destination would be $\overline{N} = \frac{1}{N}\sum_{i=1}^{N} \overline{N_i}$.

6 Performance Analysis

We assess routing robustness of AiNHOR in terms of the improved number of next-hops for primary incoming interfaces towards all destinations. Since adopting secondary interfaces for packet transportation can lead to path inflation, we also investigate the hop-count length distribution with various allowed number of next-hops.

6.1 Evaluation Setup

Network Topology We select six representative network topologies from the Rocketfuel project [28] for our evaluations. For each topology, we remove all nodes that will not contribute on routing (e.g. single out-degree node). The refined topologies are bi-connected graphs, listed in Table 1 in increasing order of their average node out-degrees. In addition, Table II shows their corresponding line graphs in increasing order of their average node out-degrees.

Table 1: Network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	50	298	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

Table 2: Transformed graphs

AS	Name	Nodes	Links	Avg. Degree
1755	Ebone(eu)	298	1432	4.80
3967	Exodus(us)	280	1372	4.90
1221	Telstra(au)	194	1030	5.30
6461	Abovenet(us)	726	5434	7.50
3257	Tiscali(eu)	564	4378	7.76
1239	Sprint(us)	1882	25988	13.81

Link weight assignments The ECMP and LFA bases its path calculation on link weights. To obtain realistic link weight settings, we implement local search heuristic [15] to optimize link load objective function under traffic matrix generated by the gravity model [27]. For AS1239, we, however, use unit link weights, because the local search heuristic does not scale to a topology of this size. Note that permutation routing of links will work with any link weight settings. We use this approach to show the performance with "typical" link weights.

Comparison We compare our AiNHOR with shortest path based routings, ECMP and LFA, and recent solution ANHOR-SP. Comparisons to other interface based routing methods (e.g. FIR [3], NISR[24] and ESCAPE [23]) are less relevant because they do not fulfill the four properties stated in our introduction.

We evaluate robustness of all mentioned routing methods in terms of links, instead of nodes. Accordingly, for given incoming interface $(* \rightarrow i)$, j is called the primary next-hop of $(* \rightarrow i)$ if $(i \rightarrow j)$ is on the SPT towards d. Otherwise, j is called secondary next-hop and $(i \rightarrow j)$ is a secondary link towards d.

6.2 Robustness Evaluation

Fig. 5 shows fractions of primary interfaces with at least two next-hops under four methods. We observe that the fractions of AiNHOR vary slightly across six topologies and are above 97%. Especially, AiNHOR achieves 100% primary interfaces with two next-hops in AS6461 and AS3257. The good results come from properties of interface based routing and its operation on the bi-connected graph. That is a node with at least two outgoing interfaces may have the chance to installed two loop-free next-hops.

Obviously, AiNHOR gives a clear improvement over LFA and ANHOR-SP. Fig. 5 shows that LFA and ANHOR-SP protect only from 21% to 67% and from 29% to 81% primary interfaces, respectively, in six tested ISP topologies.

In practice, there are good reasons to limit the number of next-hops that are installed in the forwarding tables at each interface for a particular destination. Installing more next-hops will not give much benefit with respect to robustness or load-balancing. It will, however, require more fast memory in the forwarding table, and may lead to the unnecessary inclusion of paths that are much longer than the shortest path. Therefore, those interfaces with high number of next-hops will likely have some of their next-hops involved in packet forwarding. That leads to waste many network resources when next-hop distribution resulting from a routing are more spread out.

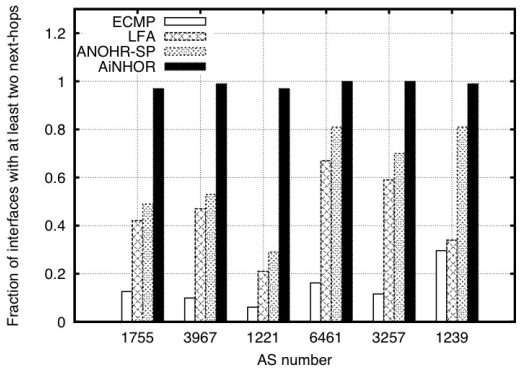
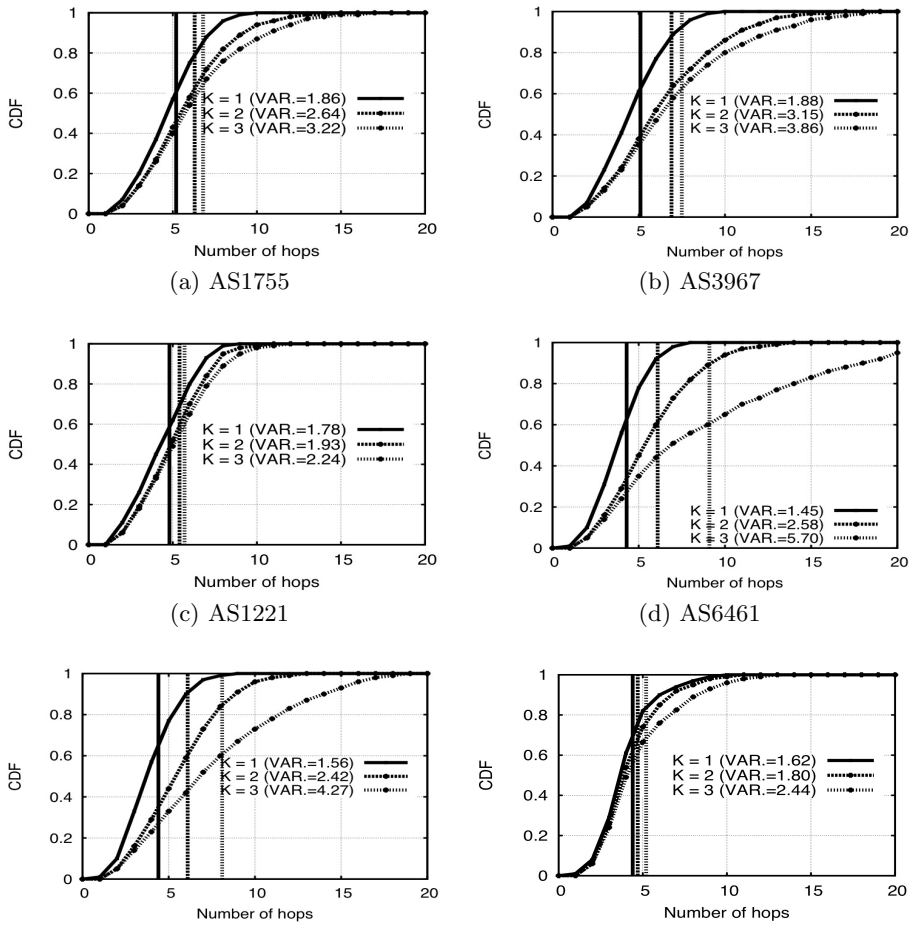


Fig. 5: Fraction of pI-D pairs with two routing options.

6.3 Path length distribution

AiNHOR has significantly improved the multipath capability that allows load balancing and increased fault-tolerance. However, adopting secondary interfaces for packet forwarding possibly leads to high path inflation which can increase

traffic load over non-shortest paths. For that reason, we investigate the distribution of path length for different values of the maximum number of next-hops for each interface, denoted by K. Note that we have measured the *longest* path corresponding to K for each source-destination pair. In other words, we show the *upper bound* of the path length for each S-D pair while packets might travel on much shorter paths in practice (where there is not the case that all primary links from the source node to the destination simultaneously fail).



(a) AS1755 (b) AS3967

(c) AS1221 (d) AS6461

(e) AS3257 (f) AS1239

Fig. 6: Path length distribution and average path length in hops

Fig. 6 shows the distribution of path lengths in hop for K =1, 2 and 3. For $K = 1$, all paths from source nodes to destinations are shortest paths. The average shortest path length is the left-most vertical line in each sub-figure. In-

creasing K to 2 and 3 will allow more longer paths and therefore shift the average upper bounds of path lengths (vertical lines) to the right. Of all topologies, those upper bounds only increase from 7.5% (AS1239) to 40% (AS6461) for $K = 2$ and up to 100% (AS6461) for $K = 3$. In practice, those path lengths are still comparable to those of shortest path routing.

6.4 Time complexity

The complexity of our proposed algorithms depends on the number of nodes, links and the length of the permutation for each destination. For all tested topologies, the average length of permutations for N sources towards N destinations, $\overline{N}$, equals half of the total number of nodes ($\overline{N} = 0.5 \times N$).

Fig. 7 shows the relative running time of AiNHOR method to ECMP across six topologies. The AS topologies are listed in an increasing order of node degrees. The results are achieved with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. For the first five topologies, the relative running times are less than ten times. The result for AS1239 is getting a bit worse, by up to 16 times. Explosion in both number of nodes and number of directed links in the transformed graph contribute in such increase. However, it is still computationally feasible.

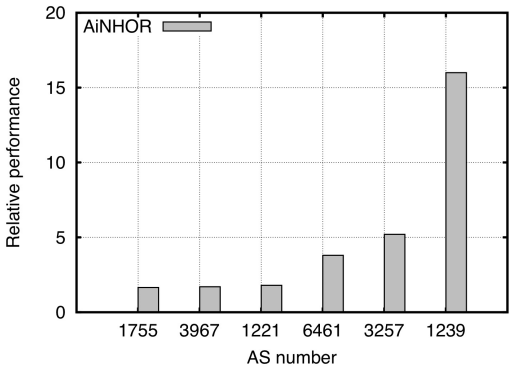


Fig. 7: Running time

7 Related Work

Many recent solutions proposed in literature seek to increase the protection coverage with different approaches. We categorized those proposals into two main groups corresponding to two forwarding methods: the traditional IP forwarding and the interface-specific forwarding. Examples of the former are Tunnels [19], Not-via [5], O2 [16], PR [14], ANHOR-SP [21], MRC [6] and IDAGs [26]. The latter includes FIR [3], ESCAP [23], NISR [24] and LFIR [20].

Tunnels [19] and Not-via [5] suggest that IP packets should be encapsulated at the node adjacent to the failure. Those IP-in-IP packets then are tunneled to an unaffected node where they are forwarded to the destination. Promising 100% single failure coverage, the tunnel technique obviously requires high overheads in tunnel signalling.

O2 [16] and PR [14] are two routings designed for the centralized routing system. Both O2 and PR use the concept of "joker" links in which both directions of a link are used for mutual backups. O2 seeks to maximize the number of nodes with two next-hops while PR allows nodes to adopt *at least* two next-hops. In addition, PR is only suitable for small scale networks due to its high complexity. ANHOR-SP [21] introduces the concept of permutation routings and proposes a generic framework, from which various routing objectives for the traditional IP network can be realized. ANHOR-SP is designed for the distributed routing system due to its low complexity. ANHOR-SP, however, provides a coverage of single link faults which is significantly lower than AiNHOR.

MRC [6] and IDAGs [26] use multiple routing tables that cover all possible failures. Upon detecting a failure on the connected link, the affected node selects another routing table to avoid network interruption and marks the routing table index in its packets. Other node will examine such index in incoming packets and will select the same configuration.

FIR [3] first introduces the interface-specific forwarding concept and proposes the shortest path based algorithm to construct forwarding tables for incoming interfaces. FIR offers 100% single link fault coverage while a later version [25] provides full coverage of single node faults. ESCAPE [23] and NISR [24] share the same idea with FIR except that routing options are not necessarily bound to shortest paths. Those papers compute back-up ports for each affected router by solving an integer linear programming problem involving single component fault situations. Those methods, however, may produce forwarding loops when multiple failures occur [20].

8 Conclusion

We have presented the permutation routing of interfaces as a method to increase robustness for IP networks with interface-specific forwarding. Our routing method is loop-free, does not introduce additional overheads for IP packets and works with the standard link state routing protocols such as OSPF or IS-IS. Our method aims to approximate the optimality of the survivability by installing as many primary interface destination pairs with at least two next-hops as possible.

We have evaluated permutation routing of interfaces with simulations on six ISP topologies. The results show that permutation routings of interfaces offer protection coverage above 97% with realistic link weight settings.

References

1. Garcia-Lunes-Aceves, J. J.: Loop-free routing using diffusing computations. IEEE Trans. on Networking, 1 (1), pp. 130–141 (1993)

2. Vutukury, S., Garcia-Luna-Aceves, J. J.: MDVA: A Distance-Vector Multipath Routing Protocol. In: IEEE INFOCOM, pp. 557-564 (2001)
3. Nelakuditi, S.,Lee, S., Yu, Y., Zhang, Z-L., Chuah, C.-N.: Fast local rerouting for handling transient link failures. IEEE Trans. on Networking, 15, pp. 359–372 (2007)
4. Atlas, A., Zinin, A.: RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates. (Sep. 2008)
5. Shand, M., Bryant, S., Previdi, S.: IP Fast Reroute Using Not-via Addresses. Internet-Draft (work in progress, expired in Dec. 2012).
6. Kvalbein, A., Hansen, A.F., Čičic, T., Gjessing S., Lysne, O.: Multiple routing configurations for fast IP network recovery. IEEE Trans. on Networking, 17 (2), (2009)
7. Elhourani, T., Ramasubramanian, S., Kvalbein, A.: Enhancing Shortest Path Routing for Resilience and Load Balancing. In: ICC, pp. 1–6 (2011)
8. Francois, P., Bryant, S., Decraene, B., Horneffer, M.: LFA applicability in SP networks. Internet-Draft (work in progress, expired in Jul. 2012)
9. Nakano, K., Olariu, S., Zomaya, A.Y.: Energy-efficient permutation routing in radio networks. IEEE Trans. on Parallel and Distributed Systems, 12 (6), (2001)
10. Liang, X., Shen, X: Permutation Routing in All-Optical Product Networks. IEEE Trans. on Circuits and Systems, 49 (4), pp. 533–538 (2002)
11. Xu, D., Chiang, M., Rexford, J.: DEFT: Distributed Exponentially-Weighted Flow Splitting. In: IEEE INFOCOM, pp. 71-79 (2007)
12. Kvalbein, A., Dovrolis, C., Muthu, C.: Multipath load-adaptive routing: putting the emphasis on robustness and simplicity. In: IEEE ICNP, pp. 203–212 (2009)
13. Cormen, T. H., et al.: Introduction to Algorithms. MIT Press, ISBN 0-262-03293-7
14. Kwong, K.-W., Gao L., Gurin, R., Zhang, Z.-L.: On the feasibility and efficacy of protection routing in IP networks. In: IEEE INFOCOM, pp. 1543–1556 (2010)
15. Fortz, B., Thorup, M.: Internet traffic engineering by optimizing OSPF weights. In: IEEE INFOCOM, pp. 519-528 (2000)
16. Schollmeier, G., Charzinski, J., Kirstadter, A.: Improving the resilience in IP networks. In: HPSR Workshop, pp. 91-96 (2003)
17. Gjoka, M., Ram, V.,Yang, X.: Evaluation of IP Fast Reroute Proposals. In: IEEE COMSWARE, pp. 1-8 (2007)
18. Ohara, Y., Imahori, S., Meter, R. V.: MARA: Maximum Alternative Routing Algorithm. In: IEEE INFOCOM, pp. 298-306 (2009)
19. Bryant, S., Filsfils, C., Previd, S., Shand M.: IP Fast Reroute Using Tunnels. In: IETF Internet Draft (expired in May 2008)
20. Enyedi G., Rétvári, G.: A Loop-Free Interface-Based Fast Reroute Technique. In: NGI, pp.39-44 (2008)
21. Vo, H. Q., Lysne, O., Kvalbein, A.: Permutation Routing for Increased Robustness in IP Networks. In: Networking 2012, pp. 217-231 (2012)
22. Druken, B., K.: Line graphs and flow nets. In: SDSU Theses and Dissertations (2010)
23. Xi, K., Chao, H., J.:IP fast rerouting for single-link/node failure recovery. In: Broadnets, pp. 142-151 (2007)
24. Lee, S., S., W., Tseng, P.-K., Chen, A., Wu, C.-S.: Non-Weighted Interface Specific Routing for Load-Balanced Fast Local Protection in IP Networks. In: IEEE ICC, pp.1-6 (2011)
25. Zhong, Z., Nelakuditi, S., Yu, Y., Lee, S., Wang, J., Chuah, C.-N.: Failure inferencing based fast rerouting for handling transient link and node failures. In: IEEE Global Internet (2005)
26. Cho, S., Elhourani, T., Ramasubramanian, S.: Independent directed acyclic graphs for resilient multipath routing. In: IEEE/ACM Trans. Networking (2012)
27. Nccui, A., Bhattacharyya, S., Taft, N., Diot, C.: IGP link weight assignment for operational Tier-1 backbones. IEEE Trans. on Networking, 15, pp. 789-802 (2007)
28. Rocketfuel topology mapping. WWW. http://www.cs.washington.edu