Permutation Routing for Increased Robustness in IP Networks*

Hung Quoc Vo, Olav Lysne, and Amund Kvalbein

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

Abstract. We present Permutation Routing as a method for increased robustness in IP networks with traditional hop-by-hop forwarding. Permutation Routing treats routers involved in traffic forwarding as a sequence of resources, and creates permutations of these resources that give several forwarding options. We introduce Permutation Routing as a concept, and use it to create routings where we seek to maximize single link fault coverage. Analogous to the IETF standardized Loop-Free Alternate (LFA), Permutation Routing can easily be implemented for OSPF or IS-IS networks to augment existing ECMP forwarding with additional loop-free forwarding entries for improved load balancing or fault tolerance. Our evaluations show that Permutation Routing can increase single link fault coverage by up to 28% compared to LFA in inferred network topologies.

Keywords: IP networks, Multipath Routing, Resilience, Fault-tolerant.

1 Introduction

The last few years have witnessed the adoption of the Internet as the preferred transport medium for services of critical importance for business and individuals. In particular, an increasing number of time-critical services such as trading systems, remote monitoring and control systems, telephony and video conferencing place strong demands on timely recovery from failures. For these applications, even short outages in the order of a few seconds will cause severe problems or impede the user experience. This has fostered the development of a number of proposals for more robust routing protocols, which are able to continue packet forwarding immediately after a component failure, without the need for a protocol re-convergence. Such solutions add robustness either by changing the routing protocol so that it installs more than one next-hop towards a destination in the forwarding table [1][2][3], or by adding backup next-hops a posteriori to the forwarding entries found by a standard shortest path routing protocol [4][5][6].

* Permutation Routing term is used in the literature with slightly different meanings, all related to the act of rearranging network objects (e.g. network devices or network devices and theirs associated radio channels) on which packets are routed from a source to a destination [9] [10].

R. Bestak et al. (Eds.): NETWORKING 2012, Part I, LNCS 7289, pp. 217–231, 2012.
© IFIP International Federation for Information Processing 2012

Unfortunately, few of these solutions have seen widespread deployment, due to added complexity or incompatibility with existing routing protocols.

Installing more than one route to a destination has obvious advantages with respect to increased robustness. First, it provides alternative routes that are readily available after a link failure. Second, it provides several paths for load-balancing, which can be used to absorb short-lived spikes in traffic demand and thus avoid congestion. Deployed intradomain routing protocols such as OSPF and IS-IS have traditionally installed only a single shortest path to each destination, or multiple equal-cost shortest paths with the ECMP extension. ECMP allows some degree of multipath routing, but can only give more than one forwarding next-hop to a limited subset of all source-destination (S-D) pairs in a network. Recently, a mechanism called Loop-Free Alternates (LFA) [4] has become available in some routers. LFA allows routers to install alternate forwarding entries that can be used as backup if the primary next-hop fails. LFA improves robustness compared to ECMP, but there will still be many S-D pairs with only a single path (next-hop) available [8].

This paper presents Permutation Routing as a novel and flexible approach for calculating multiple loop-free next-hops in networks with traditional hop-by-hop forwarding. Permutation Routing is based on the observation that routing in any network consists of using a set of resources (links and nodes) *in sequence*. A routing strategy can therefore be expressed as a permutation of the nodes that are involved in traffic forwarding to a destination. Routing will be loop-free as long as traffic can only be forwarded in one direction with respect to the node ordering in this permutation. The main focus in this paper is to use Permutation Routing to create a routing that maximizes the single link fault coverage. Other routing strategies based on permutations of *links* in networks with interface-specific forwarding [3] can also be developed.

In this paper, we provide a simple backtracking algorithm that constructs a permutation of routers for each destination, and a simple forwarding rule that allows us to generate forwarding tables based on the permutations. The properties of the resulting routing are determined by the constraints used at each step in the permutation construction. The input to the construction algorithm is the topology information that is collected by a link state routing protocol, and hence no new control plane signalling is needed with Permutation Routing.

Importantly, Permutation Routing can easily be integrated with existing intradomain routing protocols, and can be used to augment the shortest path routing tables with additional forwarding entries. We show how the constraints in the permutation construction can be designed so that the resulting routing is compatible with normal shortest path routing, while still offering significantly more forwarding options than the existing LFA. With multiple loop-free alternates for a given primary next-hop, OSPF or IS-IS may employ some of them as unequal-cost primary paths and the rest as back-up paths. In the case of multiple primary paths, packets can be distributed evenly among paths or with more intelligent load balancing methods [11][12]. In this paper we 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 organized as follows. Section 2 introduces the notion of Next-Hop Optimal Routing (NHOR), which we will use as our main design objective. Section 3 introduces Permutation Routing and the relationship between a permutation of routers and the corresponding routing graph. Section 4 describes a generic algorithm to construct permutations. Section 5 describes a specific construction algorithm that approximates NHOR, and a modified version that is also compatible with standard shortest path routing. Section 6 shows simulation results evaluating the path diversity and the computational complexity of Permutation Routing. Section 7 reviews related work. Section 8 concludes the paper.

2 Next-Hop Optimal Routing

Before introducing Next-Hop Optimal Routing as our objective function for robust routing, let us introduce some vocabulary that we will use in the rest of this paper. By a *routing*, we refer to the assignment of a set of next-hops for each destination node in each source node. We do not distinguish traffic from different sources, and hence traffic transiting a node is treated in the same way as traffic originated at the node. 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. A routing where all links are included in each DAG is referred to as a *full connectivity routing*. With a given network topology, many different high or full connectivity routings can normally be constructed. However, they will have different properties with respect to failure recovery and load balancing.

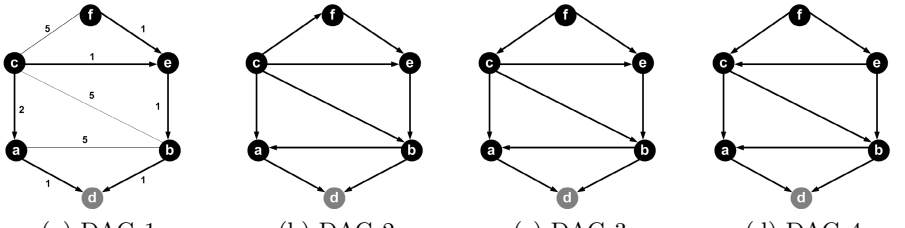

Fig. 1. A network topology with different DAGs

Fig. 1 shows a simple network topology, with 4 different DAGs for the destination node d. In Fig. 1a, DAG-1 is given by shortest path routing with ECMP using the link weights indicated in the figure. Node c can split its traffic over two next-hops, while the other nodes have only a single next-hop towards d. Links (a, b), (c, b) and (c, f) are left idle, and are neither used for backup or load balancing. The DAGs in Fig. 1b, Fig. 1c and Fig. 1d are all full-connectivity

routing graphs, where all links can be used for packet forwarding. They differ, however, in their distributions of next-hops. In DAG-2, there are three nodes (a, e and f) that have only a single next-hop towards d. DAG-3 on the other hand, has only two such nodes (a and e). DAG-2 and DAG-3 are both compatible with shortest path routing, because they contain all directed links of DAG-1. DAG-4 is not compatible with shortest path routing: by changing the direction of the link (c, e), the number of nodes with a single next-hop has been reduced to one (the minimum value).

To maximize the single link fault coverage and load-balancing capabilities of a network, it is important that there is more than one next-hop available for as many S-D pairs as possible. This leads us to the following optimization criterion for *Next-Hop Optimal Routing* (NHOR):

Definition 1. *An NHOR is a full-connectivity routing that maximizes number of S-D pairs that have* at least *two next-hops towards a destination.*

As illustrated by the example above, an NHOR will not always be compatible with shortest path routing. For that reason, we also define the *Shortest-Path compatible NHOR* (NHOR-SP):

Definition 2. *An NHOR-SP is a full-connectivity routing that maximizes number of S-D pairs that have* at least *two next-hops while containing the DAG calculated by a shortest path algorithm.*

3 Permutation Routing

We consider a network modeled as a connected graph $G = (V, E)$ where V is a set of nodes and $E \subseteq V \times V$ is the set of links (edges) and its topology. A connected link from node i to node j is denoted by (i, j).

Without loss of generality, we look at the assignment of next-hops for each destination individually. For a destination $d \in V$, let $R_d = (V, E_d)$ be a routing function for packets destined to destination d, where E_d is a set of directed links constructed on E. In R_d, node j is called a *next-hop* of node i if there exists a directed link between node i and node j, denoted by $(i \rightarrow j)$. For a loop-free routing, R_d must be a DAG rooted at destination d.

The routing function R_d contains all valid paths to d, and each path can be considered as a sequence of nodes from a specific source to d. At each node, packet forwarding is the process of sending packets to a next-hop in such a sequence. In the rest of this section, we describe how Permutation Routing can be used as a tool to find such sequences with the goal of realizing NHOR.

Definition 3. *For a given network topology $G = (V, E)$, a permutation P of nodes is an arrangement of all nodes in V 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 4. *A permutation P is a* routing permutation for R_d *if and only 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, the destination node d will always be the first node in a routing permutation for R_d. Nodes further out in the routing permutation will be topologically farther from the destination.

Lemma 1. *Any loop-free routing function R_d can* always *be represented by a routing permutation in which d is at the left-most position.*

Proof. A loop-free routing function $R_d = (V, E_d)$ 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 $d \in V$ 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 can be more than one routing permutation for one routing function R_d. Starting with a routing permutation P, another routing permutation P' can be generated by swapping *two consecutive* nodes that are not connected by a directed link to each other. For instance, both permutations $\{d\ a\ b\ e\ c\ f\}$ and $\{d\ b\ a\ e\ c\ f\}$ are routing permutations for DAG-1 according to Def. 4.

In the reverse process, routing tables can be generated from a routing permutation, 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 topological neighbors of a node that occur before it in the routing permutation are installed as next-hops. Note that this forwarding rule will result in a full connectivity routing, where all links in the topology are potentially used for traffic forwarding to all destinations. 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.

With the given forwarding rule, different routing permutations will result in routing functions with different robustness characteristics. Our goal is to find a routing permutation that can realize NHOR. This problem is believed to be NP-hard. In the next section, we present an algorithm that produces routing permutations that approximate NHOR.

4 Generating Routing Permutations

This section introduces a generic method for constructing routing permutations. The construction method is based on a backtracking algorithm, and can be used to construct routing permutations with different optimization objectives. In the next section, we specify the constraints that are used to approximate NHOR.

We reconsider a topology $G = (V, E)$ of N nodes ($|V| = N$), all of which are uniquely identified by a number from 1 to N. 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

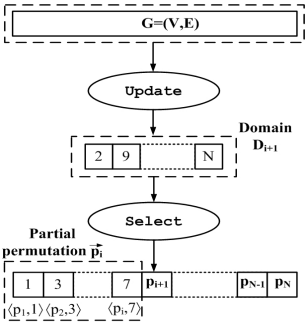

Fig. 2. Basic assignment procedure for variable p_{i+1}

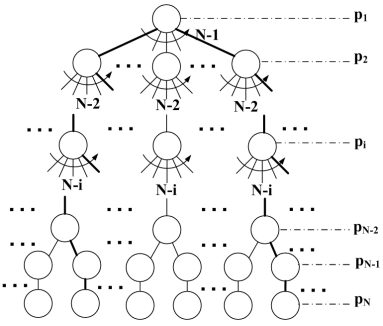

Fig. 3. Search space for routing permutation problem

$D = \{D_1, D_2, \ldots, D_N\}$. 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.

A routing permutation P is constructed by successively assigning a node $u \in D_i$ to each variable $p_i \in P$. Such assignment is said to be valid if it satisfies a specific *constraint function* $C(u)$ which is defined to realize the selected routing objective. Fig. 2 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 to obtain the routing permutation P. The algorithm calls function `Select` (with built-in constraint function $C(u)$) which goes through D_i to find a valid node for the current variable p_i. If `Select` succeeds in finding a valid assignment, the algorithm calls function `Update` to generate domain D_{i+1} and proceeds to next variable p_{i+1}. Otherwise, a backtrack step will be executed to revisit the variable p_{i-1}. 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 $C(u)$.

If the constraint function C allows it, the backtracking algorithm will find *one* routing permutation P among all possible solutions by searching through the *search space* shaped by the number of variables in P and their domains of values. In a naïve implementation, the domain for variable p_{i+1} consists of $(N - i)$ nodes that have not been placed in $\vec{p_i}$. Based on that observation, the search space S of the permutation assignment problem has a form of a tree of depth N rooted at the initial state $\langle p_1, d \rangle$ as illustrated in Fig. 3. Solutions $\vec{p}_N$ are located at its leaves. Two connected states in the search space refer to two instances of p_i and p_{i+1}. Assume that t operations are needed on average to move from state p_i to state p_{i+1}. The complexity in the best case when we do not need any backtrack

step (backtrack-free) is $O(t \times N)$. In other extreme, if there is only one solution and we always make the "wrong" choice, the complexity would be $O(t \times N!)$.

The naïve implementation described above results in high computational complexity, and is only feasible for small topologies. Hence, it is important to guide the search to avoid exploring the entire search space. In the next section, we use two main mechanisms to reduce the search space:

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

5 Next-Hop Optimal Routings

Next, we apply the framework described above to construct two high robustness routings that approximate NHOR and NHOR-SP as defined in Section 2. We call them Approximate NHOR and Approximate NHOR-SP.

5.1 Approximate NHOR (ANHOR)

With Permutation Routing using greedy forwarding, a node in p_i $(i > 2)$ has at least two next-hops if it has at least two topological neighbors that occur before it in the routing permutation. The aim of the ANHOR algorithm is to maximize the number of nodes where this is the case.

The partial routing permutation $\vec{p_i}$ represents a loop-free routing sub-graph towards destination d, denoted by $R_d^i = (V(\vec{p_i}), E_d(\vec{p_i}))$ where $V(\vec{p_i})$ is the set of i nodes in $\vec{p_i}$ and $E_d(\vec{p_i})$ is the set of directed edges formed by applying the greedy forwarding rule defined in section 3 on $\vec{p_i}$. To achieve a high robustness routing, the node selected for variable p_{i+1} to form the partial routing permutation $\vec{p}_{i+1}$ should be the node with the maximum number of topological neighbors already placed in $\vec{p_i}$. Correspondingly, the number of directed edges of the routing sub-graph formed by the partial routing permutation $\vec{p}_{i+1}$, resulted from the assignment $\langle p_{i+1}, u\rangle$, must be maximized:

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

For a more efficient implementation, we maintain a counter $c[u]$ 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 the constraint function $C_{ANHOR}(u)$ to realize the expression (1) as follow:

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

The constraint function $C_{ANHOR}(u)$ implies that the domain D_{i+1} includes all nodes that have *at least* one topological neighbor in $\vec{p_i}$. The domain is, therefore, updated following the recursive relation:

$$D_{i+1} = D_i \cup \{\ v \in V \mid (u, v) \in E\} \setminus \{u\} \tag{2}$$

where u is the node that has been assigned to variable p_i in the i-th assignment.

The computational complexity of ANHOR 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. *Constraint function $C_{ANHOR}(u)$ gives a backtrack-free algorithm for all connected input topologies.*

Proof. The proof is by contradiction. Assume that the algorithm with the constraint function $C_{ANHOR}(u)$ is not backtrack-free. This means that constraint function returns False for all $u \in D_{i+1}$ at some iteration. That can not happen because D_{i+1} can never be empty in a connected topology before all nodes have been placed in the permutation and all nodes in domain D_{i+1} always have at least one next-hop in $\vec{p}_i$.

Given the backtrack-free property of our algorithm, the complexity of calculating a permutation for *each* destination is $O(|E| + N \times |D|)$, where $|D|$ denotes the average size of the domain. Typically, $|D|$ depends solely on the average node degree of the network topology. In dense topologies, the total complexity of calculating routing permutations for *all* destinations can approach $O(N^3)$. The backtrack-free property also gives a low memory consumption because it does not need to store temporary partial routing permutations.

5.2 Approximate NHOR-SP (ANHOR-SP)

Let $R_d^{SP} = (V, E_d^{SP})$ denote the shortest path tree towards destination d. A routing permutation P whose routing function $R_d = (V, E_d)$ is an ANHOR-SP if R_d satisfies two constraints in following order:

1. R_d contains R_d^{SP}, meaning all routing choices for any node in R_d^{SP} are also valid routing choices for the same node in R_d.
2. ANHOR-SP uses the same selection criterion as ANHOR in order to maximize the number of S-D pairs with at least two next-hops.

The construction of such a routing permutation P is based on the assignment procedure described in Fig. 2. To this end, the shortest path compatibility constraint is implemented in function `Update` to limit the size of domain D_{i+1} for variable p_{i+1} and the connectivity constraint will be formalized by a constraint function $C_{ANHOR-SP}(u)$ and realized in function `Select`. Clearly, the constraint function $C_{ANHOR-SP}(u)$ is identical to $C_{ANHOR}(u)$.

The next node in routing permutation P is selected among nodes that have *all* their shortest path next hops towards d already placed in $\vec{p}_i$. Formally, let $R_d^{SP,i} = (V(\vec{p}_i), E_d^{SP}(\vec{p}_i))$ be the shortest path tree of $V(\vec{p}_i)$ nodes and $R_d^{SP,i} \subseteq R_d^{SP}$. The domain D_{i+1} for variable p_{i+1} includes all nodes u such that the assignment $\langle p_{i+1}, u\rangle$, resulting in $\vec{p}_{i+1}$, fulfills:

$$R_d^{SP,i+1} \subseteq R_d^{SP}$$

Let $c_{sp}[v]$ be the number of shortest path next-hops placed in $\vec{p_i}$ and $n_{sp}[v]$ be the total number of shortest path next-hop that can be calculated from R_d^{SP} of node v. The domain D_{i+1} for variable p_{i+1} follows the recursive relation:

$$D_{i+1} = D_i \cup \{\ v \in V \mid c_{sp}[v] = n_{sp}[v]\ \} \setminus \{u\} \tag{3}$$

where u is the node that has been assigned to variable p_i.

Proposition 2. *Constraint function $C_{ANHOR-SP}(u)$ gives backtrack-free algorithm for all connected input topologies.*

Proof. According to Proposition 1, constraint function $C_{ANHOR-SP}(u)$ always returns True unless D_i is empty. We show here that D_i can never be empty before all nodes have been placed in the permutation. If D_i is empty, there is no node that have all its shortest path descendants in $\vec{p_i}$. In other words, we can follow the shortest path DAG R_d^{SP} 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 R_d^{SP} is connected and loop-free, any path along the shortest path DAG will eventually reach the destination, which is the first node that was placed in the permutation.

The computational complexity of ANHOR-SP towards one destination includes two parts: the shortest path calculation using Dijkstra's algorithm and routing permutation construction. Due to the property of backtrack-freedom, with sparse topologies the complexity of second part towards *one* destination would be $O(|E| + |E_d^{SP}| + N \times |D|)$ where $|D|$ denotes the average size of the domain. In dense topologies, the total complexity of calculating routing permutations for *all* destinations can approach $O(N^3)$.

With a low computational complexity, ANHOR-SP can be implemented on a per-router basis in OSPF or IS-IS networks. To ensure a consistent permutation routing in the entire network, constraint function $C_{ANHOR-SP}(u)$ must return the same node in each assignment among possible selections. We can break that tie by letting the highest router ID[1] be picked.

6 Performance Evaluation

We evaluate the performance of the proposed algorithms by measuring how well they realize NHOR as defined in Def. 1. Since multipath routing leads to path inflation, we also measure the path length distribution. ANHOR and ANHOR-SP are compared to standard shortest path routing with ECMP, and to LFA. Comparisons to other robust routing methods [3][6] are less relevant, because of their different objectives and implementation costs.

[1] The highest router ID means the biggest number numerically. For instance, 192.168.1.2 would be higher than 172.16.3.2, and 172.16.3.2 would also be higher than 172.16.2.1.

6.1 Evaluation Setup

We select six representative network topologies from the Rocketfuel project [19] for our evaluations. The topologies are listed in Table 1 in increasing order of their average node degrees.

Table 1. Network topologies

AS	Name	Nodes	Links	Avg. Degree
1221	Telstra(au)	104	151	2.90
1755	Ebone(eu)	87	161	3.70
3967	Exodus(us)	79	147	3.72
3257	Tiscali(eu)	161	328	4.07
6461	Abovenet(us)	138	372	5.40
1239	Sprint(us)	315	972	6.17

The results for ECMP and LFA depend heavily on the link weight settings used in the topologies. To obtain realistic link weight settings, we run local search heuristic with link load objective function proposed in [15], using a traffic matrix generated by the gravity model [18]. For AS1239, we use unit link weights, because the local search heuristic described in [15] does not scale to a topology of this size. Note that Permutation Routing will work with any link weight settings. We use this approach to show the performance with "typical" link weights.

6.2 Robustness Evaluation

Maximizing Multipath Capability. Fig. 4 shows the fraction of nodes with at least two next-hops with the different routing methods. For reference, Fig. 4 also shows the fraction of nodes in each topology with a node degree larger than 1. Obviously, nodes with a degree of 1 can not have more than 1 next-hop to any destination. We observe that the multipath capability varies strongly between topologies; it is generally higher in more well-connected networks. ANHOR achieves a significant improvement over ECMP and LFA in all networks.

Note that the number of next-hops achieved with ANHOR is independent of link weight settings, while ANHOR-SP is constrained to including the shortest paths in the routing. ANHOR-SP performance is close to ANHOR, and gives a clear improvement over LFA (by up to 28% in AS1239). This shows that Permutation Routing can give a significant gain compared to existing solutions, while being compatible with shortest path routing with realistic link weight settings.

Next-Hop Distribution. Fig. 5 shows the mean and variance for the number of next-hops at each router in our 6 topologies. For increased robustness and load-balancing, it is generally good to have a high mean and a low variance in the number of next-hops. If this variance is high, it means that a few nodes have a high number of next-hops, while others might be left with only one.

Both ANHOR and ANHOR-SP produce full connectivity routings, which means that the mean number of next-hops across all S-D pairs will be equal to

half the average node degree in the topology. The mean is somewhat lower for LFA and ECMP, meaning that some links are not used for packet forwarding. The variance, however, is lower with ANHOR than with ANHOR-SP and LFA. This shows how ANHOR achieved a better (more uniform) next-hop distribution than the other routings.

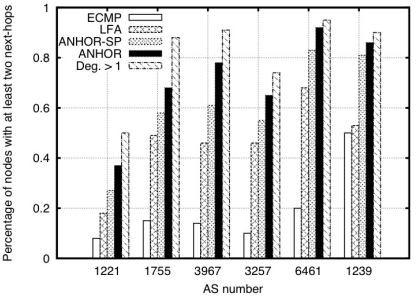

Fig. 4. Fraction of S-D pairs with at least two next-hops

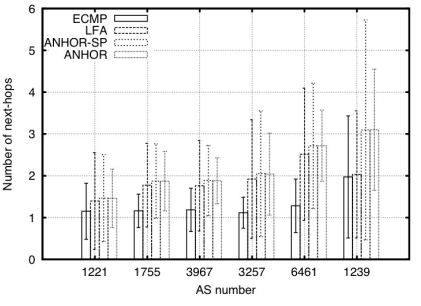

Fig. 5. Means and variances of next-hop distributions

In practice, there are good reasons to limit the number of next-hops that are installed in the forwarding table for a particular destination. Installing more than a few 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.

Hence, we look at an approach where the number of next-hops for a particular destination is limited to at most K. We define a *Routing Efficiency* coefficient, which denotes the fraction of bidirectional links that are used for traffic forwarding with a given K.

$$RE = 2 \times |E_d(K)| \ / \ |E| \tag{4}$$

where $|E_d(K)|$ is the number of directed links in the routing DAG when each node can have at most K next-hops and $|E|$ is the number of bidirectional links in the network topology. According to this definition, $0 \leq RE \leq 1$. Note that a high RE is desirable, and corresponds to a low variance in the number of next-hops achieved.

Table 2 shows the RE values for three values of K in the selected topologies. The given value is the average over all S-D pairs. We see that for all routing methods, a higher K gives a higher RE value. ANHOR and ANHOR-SP give the highest RE values, sometimes with a significant improvement over ECMP and LFA even for $K = 2$. The RE values in more well-connected topologies (AS1239) are lower than in sparse topologies. Such topologies contain a high number of nodes with a very high degree (39% nodes has their degrees greater than 15 in AS1239), and a low K will hence exclude many valid (but often unnecessary) paths.

Table 2. Routing Efficiency coefficient

		AS1221	AS1755	AS3967	AS3257	AS6461	AS1239
$K = 2$	ECMP	0.74	0.61	0.61	0.54	0.43	0.49
	LFA	0.80	0.79	0.77	0.77	0.62	0.49
	ANHOR-SP	0.86	0.84	0.85	0.76	0.67	0.58
	ANHOR	0.94	0.90	0.95	0.81	0.81	0.60
$K = 3$	ECMP	0.76	0.62	0.62	0.54	0.46	0.56
	LFA	0.86	0.90	0.87	0.82	0.77	0.57
	ANHOR-SP	0.92	0.95	0.94	0.88	0.85	0.75
	ANHOR	0.98	0.99	1.00	0.95	0.95	0.80
$K = 4$	ECMP	0.77	0.62	0.63	0.54	0.47	0.60
	LFA	0.90	0.94	0.91	0.88	0.85	0.61
	ANHOR-SP	0.96	0.99	0.97	0.93	0.93	0.85
	ANHOR	1.00	1.00	1.00	0.99	0.99	0.92

Path Stretch. High path diversity increases robustness and allows for more load balancing. However, it has a cost in terms of path inflation. Next, we look at the distribution of path lengths. We focus on path lengths in terms of hop counts, since this metric is independent of the link weight settings. Fig. 6 shows the average *path stretch* for $K = 3$ with different routings, where the length of each valid path has been normalized with the shortest path length for that S-D pair. We observe that the superior path diversity in ANHOR and ANHOR-SP comes at the cost of some path inflation, but that the average path lengths are still comparable to those of shortest path routing. The path inflation introduced with multipath routing can be ameliorated with more intelligent load balancing methods [11][12].

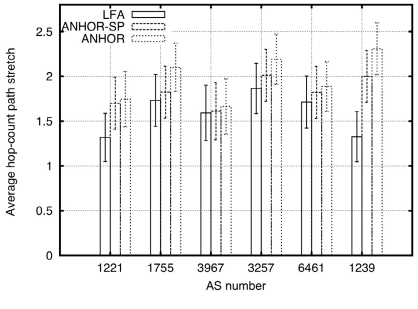

Fig. 6. Average hop-count path stretch with $K = 3$

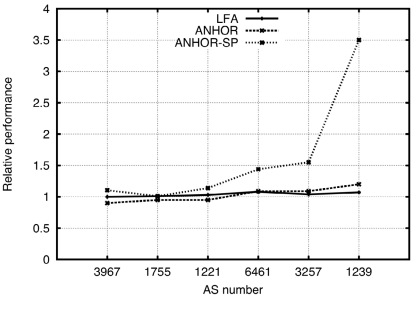

Fig. 7. Relative running time towards all destinations

Running Time and Memory Consumption. The complexity of our proposed algorithms depends on the number of nodes, links and on how efficiently the size of the candidate set can be reduced. The average size of candidate set

turns out to be approximately 5 times (AS1755) to 12 times (AS1239) higher than their corresponding average node degrees in our tested topologies.

Fig. 7 shows the relative running time of each routing method to ECMP across six topologies. The AS topologies are listed in an increasing order of number of nodes. The results are achieved with an Intel Core 2 CPU 6300 @ 1.86 GHz machine. ANHOR has a low running time that is comparable to a normal ECMP routing computation. For all destinations, the total time difference is less than 10% for all topologies. As for ANHOR-SP, calculating routing permutations for all destinations take less than four times of ECMP. Across all topologies, the memory consumption never exceeds 6 MB.

7 Related Work

This paper presents Permutation Routing as a method for increased robustness in IP networks with traditional hop-by-hop forwarding. The proposed method can be used to generate routings that give a significant boost in number of nodes that have at at least two forwarding options to a destination. Our proposal shares the same principle with many existing solutions that aim at adding robustness either by changing routing protocols to adopt more than one next-hop towards a destination in the routing table such as DUAL[1], MDVA[2], FIR[3] or adding backup next-hops a posteriori to the forwarding entries found by a standard shortest path routing protocol such as LFA[4], Not-via[5], MRC[6], SPT-DAG[7]. Unlike DUAL and MDVA, Permutation Routing bases its construction solely on readily available topology information and hence no new control plane signalling is required. In addition, Permutation Routing outperforms LFA and does not introduce overhead bits as MRC and SPT-DAG.

Recent solutions focus on centralized routing that can provide added flexibility by improving path diversity that meets different requirements. Examples are O2 [16] and Protection Routing [14]. O2 routing has a similar routing objective to Permutation Routing, but limits itself to finding only two next-hops for each node. Permutation Routing, on the other hand, allows a tunable $K \geq 1$ while still being compatible with traditional link-state routing protocols. In the same category, Protection Routing presents a two-phase heuristic to produce a routing for a given traffic demand in a centralized routing system. In phase 1, the heuristic seeks to minimize number of unprotected nodes towards a destination while minimizing the cost function given in [15]. Although Routing Permutations share the goal of minimizing the number of nodes with only one forwarding option, we prefer to evenly distribute next-hops among nodes rather than performing traffic optimization for a specific traffic demand. We believe that finding a routing that optimizes for a given load function is less important in current Internet where traffic matrix elements vary significantly with time. Instead, we aim at maximizing the available forwarding options for more intelligent load balancing methods such as [11] [12] that are more responsive to traffic variation.

Similar to our method, MARA [17] employed the concept of permutations to generate routings that can maximize alternates. MARA, however, uses a different objective function seeking to maximize the minimum number of next-hops towards a destination. It can easily be shown that in networks without parallel links between nodes, this minimum must always be 1.

8 Conclusion

We have presented Permutation Routing as an approach for calculating more robust routing while being compatible with existing links state routing protocols. The paper proposed a generic algorithm to construct routing permutations. Routings that optimize different objectives can be implemented by modifying the selection criteria that are used in the construction algorithm. The goal of Permutation Routing is to maximize the survivability in a network with traditional hop-by-hop forwarding.

We have evaluated Permutation Routing with simulations on six ISP topologies. The results show that permutation routings outperform existing multipath approaches such as ECMP and LFA in terms of robustness and path diversity. We also showed that the complexity of calculating routing permutations is comparable to that of standard link state routing.

References

1. Garcia-Lunes-Aceves, J.J.: Loop-free routing using diffusing computations. IEEE Trans. on Networking 1(1), 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, 359–372 (2007)
4. Atlas, A., Zinin, A.: RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates (September 2008)
5. Shand, M., Bryant, S., Previdi, S.: IP Fast Reroute Using Not-via Addresses. Internet-Draft (work in progress, expired in June 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 July 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), 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. Ohara, Y., Imahori, S., Meter, R.V.: MARA: Maximum Alternative Routing Algorithm. In: IEEE INFOCOM, pp. 298–306 (2009)
18. Nccui, A., Bhattacharyya, S., Taft, N., Diot, C.: IGP link weight assignment for operational Tier-1 backbones. IEEE Trans. on Networking 15, 789–802 (2007)
19. Rocketfuel topology mapping. WWW, `http://www.cs.washington.edu`