The Two Best First Hops algorithm (TBFH) An Efficient Algorithm to Enable Path Diversity in IP Newtorks

Pascal Mérindol, Pierre Francois and Olivier Bonaventure
Université catholique de Louvain (UCL)
Internet Networking Lab (INL)
Louvain la Neuve, Belgium
{pascal.merindol,pierre.francois,olivier.bonaventure}@uclouvain.be

Jean-Jacques Pansiot and Stéphane Cateloin
Université de Strasbourg (UdS)
LSIIT, UMR CNRS
Illkirch, France
{pansiot,cateloin}@unistra.fr

***Abstract*—Multipath routing allows for load balancing and fast re-routing in order to improve the reliability and the efficiency of the network. Current IP routers only support Equal Cost MultiPath (ECMP) which guarantees that the forwarding paths do not contain loops. However, ECMP provides limited path diversity. In this paper, we present an efficient algorithm that allows routers to enable more path diversity: our algorithm let each router computes at least the two best first hop distinct paths towards each destination and achieves a good tradeoff between path diversity and overhead.**

In addition, we propose a multipath routing scheme whose goal is to combine fast re-routing and load balancing loop-free routes. The low overhead of our scheme (no additional signaling messages and low complexity) and the nature of its loop-free rules allow to incrementally deploy it on current IP routers. Using actual, inferred, and generated topologies, we compare our algorithm to existing solutions.

I. Introduction

Multipath routing allows for load balancing and fast re-routing in order to enhance the network reliability and efficiency. Despite these potential benefits (e.g., [7] or [11]), most IP networks still use unipath routing protocols such as OSPF or IS-IS. With these routing protocols, the forwarding engine only reacts upon topology modifications, intentional or not, but not upon traffic variations. Dynamic multipath routing (e.g., [45], [44], or [5]) can provide services such as load balancing to reduce delays and improve throughput. Furthermore, the recovery of an IP network after failures depends on the time necessary for the convergence of the underlying routing protocol. Pre-computed alternate routes can be directly used as emergency exits without waiting for the routing protocol convergence.

In the context of intra-domain multipath routing, current IP routers only support Equal Cost MultiPath (ECMP) to enable path diversity. This feature uses a simple variant of the Dijkstra algorithm [13] where equal cost paths are inherited during the computation of the Shortest Path Tree (SPT). The optimality of sub-paths computed with ECMP ensures the loop-freeness of this approach, but restricts the number of valid forwarding paths. The advantages of ECMP are limited to cases where equal cost paths exist. Note that IGP weights can be modified to increase the level of protection of a network ([23]). However, if IGP weights are not tuned to favor the ECMP utilization, then the number of router pairs between which ECMP can use multiple equal cost paths is very low as it is the case with the Abilene network which uses link latencies as IGP weights.

There are two main approaches to avoid this limitation using multiple unequal cost paths between a pair of routers. On the one hand, source multipath forwarding schemes (e.g., [14] or [25]) can use MPLS with a signaling protocol (such as *RSVP-TE*) to establish the desired routes. On the other hand, hop by hop multipath forwarding schemes can be used to limit the signalization overhead. They do not require end-to-end signaling, packet marking or another layer of encapsulation in the data plane. However, routers must select and validate a subset of their next hops such that their distributed composition does not create forwarding loops (see [43], [44] and [46]).

In this paper we focus on this second type of schemes. We propose a distributed forwarding scheme that computes a set of loop-free routes allowing to enable fast reactions in case of failure or congestion. Our approach is incrementally deployable in the sense that it provides loop-free routes even if only a subset of routers implements our solution.

The main contribution of this paper is a multipath computation algorithm computing at least two paths towards any destination if the network is 2-edge connected. On the one hand, it has been demonstrated (see [31] for a formal description of the problem, or [4] for a measurement based evaluation with a multihoming perspective) that two forwarding alternatives are generally sufficient to strongly improve the network usage. On the other hand, for resiliency purpose, two pre-computed forwarding next hops per destination are sufficient to protect the network from single link failures. To achieve those goals in a hop by hop forwarding context, we present an algorithm called TBFH which computes the Two Best First Hop disjoint paths. To the best of our knowledge, our proposition is the lowest time complexity solution that performs this computation. The time complexity of our algorithm does not depend on the degree of the calculating router as it is generally the

case with existing methods.

In addition, we provide a general multipath forwarding scheme whose goal is to combine load balancing and fast re-routing capabilities. Finally, we also investigate and discuss several deployment issues. In particular, we focus on the multipath interaction with TCP and we also propose different levels of tradeoff between complexity and path diversity.

The remainder of the paper is organized as follows: Sec. II presents the general context of multipath hop by hop forwarding while Sec. III provides technical details of our proposition. Then, Sec. IV deals with deployment considerations. Finally, Sec. V gives an overview of the related work and Sec. VI shows the efficiency of our approach using a large variety of networks.

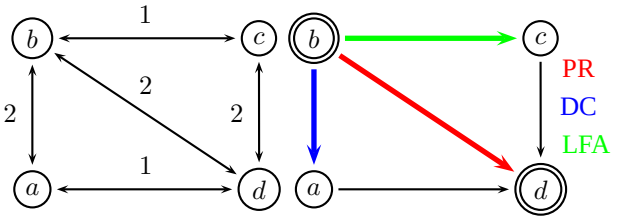
II. Hop by Hop Multipath Forwarding

In this section we describe the context related to hop by hop multipath forwarding schemes. In this context, computed paths are elementary and first hop distinct. Table I lists the graph definitions used in the paper. For the sake of clarity and without loss of generality, we do not consider the multigraph issue: a first hop is equivalent to a successor node, the next hop.

The function w assigns the weight of each edge which is flooded by the routing protocol. Paths are sorted according to an additive metric C, and we focus on shortest paths having distinct first hops. With a hop by hop forwarding perspective, only the next hop, e.g., the first hop of a computed path, is used in the forwarding plane: the distributed composition of these next hops forms forwarding paths, the routes. To distinguish equal cost paths, we consider the lexicographical order of their first hops.

Conceptually, with a hop by hop link state routing protocol using multiple unequal cost paths, two phases are necessary to ensure loop-free forwarding: a multipath computation algorithm and a validation process. The first phase computes a set of alternate paths using distinct first hops towards each destination. This phase is detailed in Sec. III while Sec. II-C provides a brief description of existing work. The second phase selects among such alternate paths those ensuring a loop-free forwarding scheme. More precisely, this phase ensures that any hop by hop composition of the validated next hops does not create forwarding loops. Sections II-A and II-B provide examples of such validation processes for fast re-routing and load balancing purposes.

In order to describe these phases more deeply, let us define two kinds of paths. A *primary* path denotes the optimal path linking a given pair of nodes (s, d). The path optimality depends on the cost C according to the link weights w and a lexicographic order to rank equal cost paths. The cost of a path is given by the metric used by C (which is usually additive and such that w takes link capacities into account). For a given pair (s, d), an *alternate* path is a path whose first hop is distinct from the first hop of the primary path $P_1(s, d)$. In the context of hop by hop forwarding, a route to a destination d is defined by its first hop: the next hop used to reach d. Although



(a) A weighted topology (b) Validation rules

Fig. 1. Alternate paths and validations rules

there may exist several forwarding paths using the same first hop n, in the path computation phase we focus on the best elementary path using n as first hop. In the remainder of the paper, as we consider a distributed forwarding plane, alternate paths are first hop distinct so that terms primary/alternate paths $P_j,\ j \geqslant 1$ refers to primary/alternate next hops $NH_j,\ j \geqslant 1$. In particular, we focus on the next hop which will be used after the convergence of the routing protocol following the failure of the primary next hop. We call such a next hop a *local post convergence* next hop: it is the first hop of the optimal alternate path.

Definition 1: A node v is a local post convergence next hop for a pair $[s, d]$ if $v = NH_2(s, d)$.

Now, let us define a safety property for hop by hop forwarding protocols guaranteeing the forwarding loop-freeness.

Definition 2: A forwarding scheme is loop-free at the router level if it always converges to a stable state such that when any router *s* forwards a packet towards any destination *d* via any validated next hop *v*, this packet never comes back to *s*.

Note that a post convergence next hop does not provide any guarantee of loop-freeness during transient periods of topology changes. Let us consider the example of the weighted topology given in Fig. 1(a). The primary path $P_1(a, d)$, denoted PR in Fig. 1(b), has a cost of $C_1(a, d) = 1$ while the local post convergence next hop $b = NH_2(a, d)$ for this pair (a, d) is the first hop of the path $P_2(a, d)$ whose cost is $C_2(a, d) = 4$. If a uses b as an alternate next hop for destination d, then a forwarding loop between a and b may appear if b also uses a as an alternative next hop for d. This figure illustrates the need for loop-free rules to enable multipath routing in IP networks.

A. Fast re-routing

Fast re-routing techniques improve network resiliency. Techniques such as loop-free alternates (*LFA*, [6]) are generally able to cover more than half of the network from single link failure (Sec. VI-C and [18] provides an evaluation of the LFA coverage in actual IP networks). The purpose of LFA is to select backup next hops in order to handle local failures with a very low overhead. When such LFA next hops exist, it allows to avoid the use of more sophisticated re-routing scheme such as MPLS-FRR [41] or Notvia [9].

In case of failure, the goal is to quickly converge towards the new stable routing state and, during this fast convergence period, select backup next hops that do not induce transient loops. Five periods determine the convergence time of a link-state routing protocol ([19]):

TABLE I
GENERAL NOTATIONS

Notations	Definitions
$G(N, E, w)$	oriented graph G with a set of nodes N, a set of edges E and a positive valuation of edges w
$e = (x, y)$	edge $e \in E$ connecting node x to node y
$succ(x)$	set of successor nodes of x
$deg(x)$	outgoing degree of node x, $deg(x) = \lvert succ(x) \rvert$
$P_j(s, d)$ $j \leqslant deg(s)$	j^{th} best elementary path linking s to d. This is the best path whose first hop is distinct from the first hop of the $j-1$ best paths
$C_j(s, d)$	cost of the elementary path $P_j(s, d)$
$NH_j(s, d)$	j^{th} best next hop computed on s towards d.This is the first hop of $P_j(s, d)$.

1- The physical layer reaction time (e.g, SDH or SONET alarms).
2- The link layer reaction time (e.g, *Hello* or BFD messages).
3- The advertisement flooding time (e.g, the exchange of Link State Advertisements messages, *LSA*).
4- The SPT re-computation time (the update of the Routing Information Base, RIB).
5- The Forwarding Information Base (FIB) update.

Techniques such as LFA allow to save the three longest periods, the LSA flooding time and the RIB and FIB updates (steps 3 to 5). Indeed, with the appropriate FIB architecture (Sec. III-A), if there exists a local pre-computed alternate next hop, it is possible to use it during transient periods of topology change. A LFA next hop v verifies:

$$C_1(v, d) - C_1(v, s) < C_1(s, d) \quad (1)$$

This rule means that v does not use s as its primary next hop to reach d. The LFA rule guarantees the absence of forwarding loops as long as there is only one link failure in the network. In Fig. 1, node b can validate node c as a LFA next hop protecting from the failure of the primary first hop (b, d). Indeed, the path (c, d) has a cost that is strictly lower than $C_1(c, b)+C_1(b, d) = 3$.

B. Load balancing

Load balancing allows for congestion avoidance: several next hops can be used simultaneously to provide routing flexibility and increase the total bandwidth of each flow. The objective is to find a set of next hops whose distributed composition does not induce forwarding loops. Stronger rules than with the re-routing issue are necessary: we need to compute loop-free next hops that can be simultaneously used.

To increase the number of forwarding alternatives compared to ECMP, the simplest rule to select a next hop v on a router s (such that $v \in succ(s)$) is the *downstream criterion* (*DC*) which can be expressed as follows:

$$C_1(v, d) < C_1(s, d) \quad (2)$$

This rule is defined in [1], used in OSPF-OMP [43] and is denoted *LFI* in [44] (with an additional property avoiding forwarding loops even in transient periods of topology changes). Gojmerac and al. [20] have shown that the path diversity achieved by this simple rule is much more important than the one obtained with ECMP (Sec. VI-C also highlights this result in actual IP networks). Note that the ECMP and the DC rules can be used both for load balancing and fast re-routing purposes.

In Fig. 1, node a is a DC next hop for the pair (b, d). Indeed, the path (a, d) has a cost strictly lower than the one of (b, d). Note that a DC next hop is also by construction a LFA next hop, but the reciprocity is not true. The LFA next hop c does not verify rule (2) for the pair (b, d).

C. Alternate path computation

The path computation method used in OSPF or IS-IS is a Shortest Path First algorithm (SPF) such as Dijkstra's algorithm. To enable the diversity of forwarding paths, it is necessary to compute alternate paths. The goal of this phase is to compute a set of elementary paths whose first hops are distinct: the candidate next hops.

In order to compute costs $C_1(v, s)$ used in rules (1) and (2) for any neighbor v and any destination d, a router can perform multiple SPT computations. This computational phase allows to avoid the exchange of messages between neighbors. The most trivial technique consists in computing the SPT of each neighbor of the root node ([6] and [33]). This kind of method is denoted kSPF in the remainder of the paper. The cost $C_1(v, s)$ used in rule (1) can be computed either thanks to a reverse SPT computation, or using the SPT rooted at v, or considering the minimum between the shortest elementary path connecting s and d which uses v as first hop and the cost $C_1(s, d)$. The main drawback of a kSPF algorithm is that it increases the time complexity of the computation phase by a multiplicative factor of $k = deg(s)$. For routers having a large degree, this factor is not negligible and the time complexity overhead can become significant for large IP networks.

Our proposal focuses on post convergence next hops which are first hops of optimal alternate paths. On the one hand, it allows to strongly reduce the time complexity required by the multipath computation phase. Indeed, the time complexity of our algorithm is lower than two SPT computations. On the other hand, focusing on post convergence next hops minimizes the path flapping in case of failure as described in Sec. IV-B. However, in some singular cases described in Sec. III-D, there does not exist post convergence next hops verifying rules (1) or (2) while there exists valid next hops. In these cases, the path diversity is not optimal compared to a kSPF algorithm applied to rules (1) or (2). In Sec. VI-C, we show that post

convergence next hops are sufficient to perform an interesting tradeoff between path diversity and computation time. Cases where there exists a valid alternate next hop while there does not exist a valid post covergence one are extremely rare.

Furthermore, our approach may allow for considering more sophisticated forwarding architectures such as the incoming interface dependent forwarding scheme described in [29]. Indeed, with the scheme developed in [29], it can be useful to know neighbor alternate costs in order to provide greater path diversity than with a forwarding scheme solely based on the destination.

III. Our Proposal

The goal of our algorithmic approach is to provide the two best first hop disjoint paths between each pair of routers if such paths exist (the network must be 2-edge connected). For that purpose, we propose a multipath computation algorithm able to compute a set of pairs $\{[C_j(s,d), NH_j(s,d)]\}\ _{\forall d \in N}$ containing at least the two best elements for each destination d. Using such a multipath computation algorithm, rule (2) becomes:

$$C_j(s,d) - w(s,v) < C_1(s,d) \tag{3}$$

If the next hop $v = NH_j(s,d)$ satisfies rule (3), then v is a valid DC next hop: the j^{th} best next hop can be used by s to reach d because it satisfies Definition 2. For fast re-routing objective, the LFA rule (1) becomes:

$$C_j(s,d) - w(s,v) - C_1(v,s) < C_1(s,d) \tag{4}$$

Rule (4) takes the term $C_1(v,s)$ into account such that a DC next hop is also a LFA next hop. However, a LFA next hop which does not verify rule (3) satisfies Definition 2 only if it used after the failure of the link $(s, NH_1(s,d))$ (and if there is no other failure in the network).

To summarize, our approach follows these three steps:

1- it uses a classic link state control plane to get topological information (LSA flooding),
2- it uses a multipath computation algorithm instead of a basic SPT to compute primary and candidate alternate next hops,
3- it uses a specific condition in order to ensure the loop-free property and thus select valid next hops.

Note that our multipath computation algorithm can be used with non local loop-free rules. We provide DC and LFA rules to illustrate two simple applications. Using IP-in-IP tunneling capabilities [8], it can be demonstrated that the protection coverage can be complete if IGP link weights are symetric and if the network is 2-edge connected.

A. Combining re-routing and load balancing

We envision a combined load balancing/fast reroute scheme using three modes of forwarding instead of a single one containing only the primary next hops set (denoted PR). In order to improve the forwarding diversity, we define three sets of next hops: the MultiPath set (MP), the Fast-ReRouting set (FRR) and the Fast Convergence set (FC).

The first mode is dedicated to load balancing (PR+MP sets): for a given destination, all next hops belonging to the MP set can be used simultaneously with the primary next hop to enable load dependent routing. The second mode (using the FRR set) is used for fast reroute when all next hops in the load balancing mode have failed, e.g., it provides loop-free next hops able to locally handle a failure outage during transient period of topology changes. During the outage, note that FRR next hops can be added to the MP set if they verify the DC rule using $C_2(s,d)$ as primary cost. The last mode can be used to improve the fast convergence (FC) in case of failure (no more valid next hop in the PR/MP and FRR modes), e.g., it allows to directly use the new best next hop but does not provide any guarantees on forwarding loop-freeness. In this paper, the MP set corresponds to the DC rule, the FRR set to the LFA rule and the FC set consists in post convergence next hops that are neither in the DC set nor in the LFA set. Note that the FC next hops set can be empty if the FRR method provides a complete coverage for a given destination.

Our scheme can be summarized this way, for a given destination d, if a failure occurs:

- locally on the primary next hop towards d: use alternate next hops instead, and trigger IGP convergence.
- locally on an alternate next hop towards d: do not use this alternate next hop anymore and trigger IGP convergence.
- remotely (a LSA is received): remove DC next hops whose alternate paths towards d contains this link and trigger IGP convergence.

In this scheme, triggering IGP convergence means recomputing primary next hops and then compute and validate alternate next hops. We assume that the FIB architecture is designed to contain multiple next hops towards a given destination d. The nature of the forwarding function is not in the scope of this paper. However, an architecture such as PIC (Prefix Independent Convergence FIB architecture which is already supported by recent Cisco Systems platforms, [17]) gives some insights about such a forwarding function considering the IGP/ BGP interaction. Such an architecture allows to activate the switchover to alternate next hops in a time that does not scale with the number of impacted destinations.

However, the use of the LFA rule without considering DC next hops may create transient forwarding loops. In order to use the load balancing mode during transient periods of topology change, PIC needs to be complemented with a forwarding function $ff(d,l) \rightarrow \{NH\}$ able to act as a filter to avoid remote failing link l. $\{NH\}$ denotes the set of remaining valid next hops towards d after the failure of l. In order to combine DC and LFA next hops without generating transient loops, the ff function needs to take remote link failures into account using LSA. If the failing link is included in an alternate path corresponding to a DC next hop towards d, this DC next hop cannot be included in $\{NH\}$. The removal of alternate paths containing the failing link allows to avoid transient loops and thus combine load balancing and fast rerouting. Note that this is also possible to simply de-activate the MP set during the

IGP convergence. Obviously, the tradeoff between the gain (combining load balancing and fast rerouting) and complexity (taking remote failures into account) of this improvement must be analysed. Indeed, the gain of enabling load balancing during the IGP convergence depends on traffic demands. Note that the functionalities and associated gains suggested in this paper do not rely on such a FIB feature.

FRR next hops are used to provide emergency exits during periods of convergence whereas MP next hops are designed to perform load balancing. A possible application of our scheme is to implicitly expose multiple forwarding paths to end hosts. Thus, at layer-4, end hosts can take advantage of the resulting distributed path diversity thanks to a path selector mechanism. This mechanism and related issues with TCP are detailed in Sec. IV-B.

B. Alternate Path Properties

In the context of hop by hop forwarding, an alternate path denotes a path whose first hop is distinct from the one of the primary path. More precisely, we are interested in a subset of alternate paths verifying the post convergence property: the optimal 1-alternate paths. Table II summarizes definitions related to our path terminology. Fig. 2 serves as a basis to understand properties of 1-alternate paths.

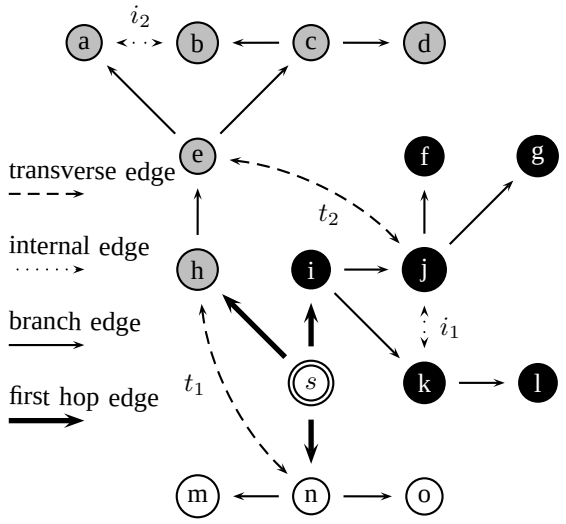


Fig. 2. *Branches*, *transverse* and *internal* edges

Given a root node s, the set of edges[1] of a graph can be partitioned into four subsets:

- Edges connecting s to first hops of primary paths.
- Edges belonging to subtrees of the SPT forming *branches*.
- *Transverse edges* connecting two distinct branches or connecting the root s and a branch without being the first hop of a primary path.
- *Internal edges* linking nodes of the same branch without belonging to this branch.

[1]We consider both directions of each edge.

The union of these four subsets captures E. A direct consequence of this decomposition is the fact that an alternate path contains at least one transverse edge. Fig.2 illustrates this edge partition in a simple graph. Note that we consider that the weights of each link is fixed to 1 such that the metric that we use is the path length in hop number. The SPT rooted at s includes three branches illustrated by three different colors (black, grey and white nodes). In this topology, there are two *transverse* edges (dashed arcs) and two *internal* edges (dotted arcs). The plain edges form the SPT. The edges i_1 and i_2 are called internal because they connect nodes belonging to the same branch, whereas edges t_1 and t_2 are transverse edges because they connect nodes belonging to different branches. A path must contain nodes of at least two different branches to be an alternate path except if the first hop n of the alternate path corresponds to a transverse edge. In this case, this means that the shortest path towards neighbor n is not the direct link so that this direct link corresponds to a transverse edge. In any case, this implies that an alternate path contains at least one transverse edge. In the hop by hop forwarding context, an alternate path is necessarily a k-alternate path for some $k \geqslant 1$.

Fig. 2 illustrates the k-alternate paths terminology. Edges (s,h), (s,i) and (s,n) are the three first hops (thick arcs) linking s to the three branches. Path (s,h,n) is a *simple alternate* path and (s,i,j,e,c) is a 1-alternate path. Path (s,n,h,e,j) is 2-alternate, it contains two transverse edges t_1 and t_2.

In practice, it means that router s uses three primary next hops for its forwarding plane: $NH_1(s) = \{h,i,n\}$. If the destination belongs to:

- $\{a,b,c,d,e,h\}$, s uses h as primary next hop.
- $\{i,j,k,l,f,g\}$, s uses i as primary next hop.
- $\{m,n,o\}$, s uses n as primary next hop.

Furthermore, having a post convergence next hop $h = NH_2(s,n)$ (the first hop of a simple alternate path) verifying rule (4) means that s can use h as a LFA next hop towards destination n. By generalizing to all nodes in $branch_n(s)$, s can use h as a LFA next hop for any destination in $\{m,n,o\}$. Indeed, there exists a 1-alternate path verifying rule (4) towards each of these destinations.

C. Optimality of 1-alternate paths

In this section, we describe the properties of alternate paths that contain only one transverse edge: 1-alternate paths. Let $\circ$ be the operator representing path concatenation and ϵ be the notation for an empty path.

Several alternate paths using the same first hop towards a given destination may exist. Considering destination d, we focus on the best path among the set of 1-alternate paths using the same first hop towards d. To refer to those paths we use the term *optimal 1-alternate path*. An optimal 1-alternate path $\mathcal{P}(s,d)$ can be decomposed as follows:

$$\mathcal{P}(s,d) = p_1 \circ (x,y) \circ p_2$$

with $p_1 \in \{P_1(s,x), \epsilon\}$, edge $(x,y) \in trans(G,s)$, and p_2 not containing any transverse edge. Note that the simple alternate

TABLE II
MULTIPATH TERMINOLOGY

Terms	Definitions
$NH_1(s)$	set of primary first hops $h \in succ(s)$ such that $\exists d \in N \mid h = NH_1(s,d)$
$branch_h(s)$	subtree of the SPT of s rooted at a neighbor $h \in NH_1(s)$
transverse edge	an edge is transverse if it connects two distinct branches $branch_h(s)$ and $branch_{h'}(s)$ or if it connects the root s and a node $n \neq h$ in a $branch_h(s)$
$trans(G,s)$	set of all transverse edges considering a root node s and a graph $G(N,E,w)$
internal edge	an edge $e=(x,y)$ is internal if it connects two nodes x and y belonging to a given $branch_h(s)$ and such that $e \notin branch_h(s)$
k-alternate path	a path is k-alternate if it contains exactly k transverse edges
simple alternate path $\in Pt(s,d)$	a 1-alternate path $(s,...,n_m,d)$ such that $(s,...,n_m) = P_1(s,n_m)$ and (n_m,d) is a transverse edge

path $p_1 \circ (x,y)$ is by definition entirely edge-disjoint from the primary path $P_1(s,d)$. The optimal 1-alternate path between s and d verifies $p_2 = P_1(y,d)$, where p_2 may contain one or several internal edges. We have the following property:

Property 1: If there exists an alternate path $\mathcal{P}'$ from s to d, then there exists a path from s to d whose cost is not greater than the one of $\mathcal{P}'$ and containing only one transverse edge.

Proof: Let $\mathcal{P} = (s,...,n_i,n_{i+1},...,d)$ be an alternate path from s to d where (n_i,n_{i+1}) is the last transverse edge of $\mathcal{P}$. Let $P_1(s,n_i)$ be the primary path from s to n_i. Then, either $P_1(s,n_i) = \epsilon$ because the edge (s,n_{i+1}) is transverse, or $P_1(s,n_i)$ is not longer than $(s,...,n_i) \in \mathcal{P}$. In both cases, there exists an alternate path $\mathcal{P}' = P_1(s,n_i) \circ (n_i,n_{i+1}) \circ (n_{i+1},...,d)$ containing only one transverse edge and such that its cost is not greater than the one of $\mathcal{P}$. ■

Figure 2 illustrates Property 1. The path $\mathcal{P} = (s,n,h,e,j)$ between s and j goes through $branch_n(s)$ and $branch_h(s)$ to reach the transverse edge t_2. This path contains two transverse edges whereas there exists a simple alternate path $\mathcal{P}' = P_1(s,e) \circ (e,j)$.

A direct consequence of Property 1 is that if there exists an alternate path $\mathcal{P}$ from s to d, then at least one of the optimal alternate paths towards d is a 1-alternate path. For a given destination, it is not possible for a k-alternate path ($k > 1$) to have a strictly lower cost than an optimal 1-alternate path. Indeed, by definition, the best simple alternate path reaching a given branch at the same node has necessarily a cost lower or equal to any other alternate path reaching this branch (Property 1). This consequence motivates the design of our algorithm: the goal is to take advantage of Property 1 by focusing on the search for optimal 1-alternate paths.

D. The TBFH Algorithm

TBFH stands for the *Two Best First Hops* algorithm. TBFH computes the primary next hop and a set of candidate next hops towards each destination. A candidate next hop is the first hop of a computed alternate path. Each candidate next hop is associated to the cost of its alternate path for further validation ensuring loop-free forwarding.

TBFH requires only one additional reduced SPF computation while it computes all local post convergence next hops. Our algorithm uses two distinct phases of computation: TBFH-1 (Alg. 1) and TBFH-2 (Alg.2). TBFH-1 performs the computation of the set of primary next hops and partitions the graph into several non connected components. These connected components are given as an input for TBFH-2 which performs a SPF algorithm on each of them and then returns a set of candidate couples (alternate next hop, alternate cost). The separation between those two phases is useful to uncouple the primary next hops computation from the computation of candidate alternate next hops. Hence, it is possible to forward packets on primary next hops while the computation and validation of alternate next hops is performed as an independent task. Without loss of generality, sets used in Alg. 1 and Alg. 2 (e.g, Tc or Tp) can be implemented through various kinds of priority queue (see Sec. III-E): lists, heaps or other optimized structures. In any cases, those structures use the destination node as a key to access their elements such that $Tc(y) = cost$ means that the best current cost towards the destination y is $cost$.

TBFH-1 decomposes the graph G into $|NH_1(s)|$ subgraphs: $G_s^h,\ \forall h \in NH_1(s)$. Each subgraph $G_s^h(N_s^h, E_s^h, w_s^h),\ h \in NH_1(s)$ has its own set of nodes and edges. The only intersection between those sets is the root node s. A subgraph G_s^h is the union of the calculating node s, the branch $branch_h(s)$ rooted at $h \in NH_1(s)$, the internal edges connecting nodes belonging to $branch_h(s)$ and a set of *virtual links* where this set represents the subset of optimal simple alternate paths $\in Pt(s,x),\ \forall x \in branch_h(s)$. A virtual link (s,x) has a specific weight function $w_s^h((s,x))$. This weight is equal to the cost of the optimal simple alternate path linking s and x. Note that none of these paths use h as first hop. Virtual links are used instead of transverse edges such that there are as many edges in E as the number of edges resulting of the union of $E_s^h,\ h \in NH_1(s)$. All other links (internal and those belonging to the SPT) have the same weight as the ones given by w. In practice, TBFH-1 partitions nodes in subsets N^h (line 37) and computes simple alternate paths towards each groups (lines 20, 30, 32 and 39). A virtual link in V^h is a triplet (a,b,c) where a is the destination node, and b and c are respectively the cost and the first hop of the optimal simple alternate path towards a ($c \neq h$). Sets H and W are used to store the optimal alternate costs and corresponding next hops until each given destination node is definitely assigned to a branch (line 37). A virtual link is definitively added (lines 30 and 39) when the destination node is marked (line 36), otherwise TBFH stores

Algorithm 1 Algorithm TBFH step 1

1: **procedure** TBFH-1($G(N, E, w), s$)
2: $Tc,\ Tp$: Set of best cost and next hops
3: T: Set of marked nodes
4: H, W: Set of virtual hops and weights
5: $\{N^h\}$: Set of the set of nodes in each $branch_h(s)$
6: $\{V^h\}$: Set of the set of virtual links for each $branch_h(s)$
7: $Tc(d) \leftarrow \infty,\ \forall d \in N$
8: $\{N^h\} \leftarrow \varnothing,\ \{V^h\} \leftarrow \varnothing$
9: $H(d) \leftarrow \infty,\ W(d) \leftarrow \infty,\ \forall d \in N$
10: $T \leftarrow \varnothing,\ Tp \leftarrow \varnothing$
11: $Tc(s) \leftarrow 0$
12: **while** $|T| < |N|$ **do**
13: Choose the node x ($x \notin T$) of minimum cost $Tc(x)$
14: $h \leftarrow Tp(x)$
15: **for** $all\ y \in succ(x)$ **do**
16: $cost = Tc(x) + w(x, y)$
17: **if** $cost < Tc(y)$ **then**
18: $tp \leftarrow Tp(y)$
19: **if** $tp \neq h \wedge tp \neq \varnothing$ **then**
20: $H(y) = tp,\ W(y) = Tc(y)$
21: **end if**
22: $Tc(y) = cost$
23: **if** $x = s$ **then**
24: $Tp(y) = y$
25: **else**
26: $Tp(y) = Tp(x)$
27: **end if**
28: **else if** $h \neq Tp(y) \wedge cost < W(y)$ **then**
29: **if** $y \in T$ **then**
30: Add virtual link $(y, cost, h)$ in V^h
31: **else**
32: $H(y) = h,\ W(y) = cost$
33: **end if**
34: **end if**
35: **end for**
36: Put x in T
37: Put x in N^h
38: **if** $W(x) < \infty$ **then**
39: Add the virtual link $(x, W(x), H(x))$ in V^h
40: **end if**
41: **end while**
42: Return $\{N^h\}$ and $\{V^h\}$ for TBFH-2
43: Return Tp as the set of primary next hops
44: **end procedure**

Algorithm 2 Algorithm TBFH step 2

1: **procedure** TBFH-2($G(N, E, w),\ \{N^h\},\ \{V^h\},\ s$)
2: A: Set of couples (alternate next hop, alternate cost)
3: **for** all $N^h \in \{N^h\}$ **do**
4: $Tc,\ Tp$: Set of best costs and next hops
5: T: Set of marked nodes
6: $Tc(d) \leftarrow \infty,\ \forall d \in N^h$
7: $T \leftarrow \varnothing,\ Tp \leftarrow \varnothing$
8: **for** each triplet $(a, b, c) \in V^h$ **do**
9: $Tc(a) = b,\ Tp(a) = c, A(a) = (b, c)$
10: **end for**
11: Put s in T
12: **while** $|T| < |N^h|$ **do**
13: Choose the node x of minimum cost $Tc(x)$
14: **for** $all\ y \in succ(x)$ **do**
15: **if** $y \in N^h \wedge Tc(x) + w(x, y) < Tc(y)$ **then**
16: $Tc(y) = Tc(x) + w(x, y), Tp(y) = Tp(x)$
17: $A(y) = (Tp(y), Tc(y))$
18: **end if**
19: **end for**
20: Put x in T
21: **end while**
22: **end for**
23: Return A as the set of candidate next hops
24: **end procedure**

temporary values in sets H and W (lines 20 and 32). Line 13 performs the search for the minimum cost while line 22 performs the best cost update or insertion.

During a second phase, TBFH-2 performs a SPF algorithm on each subgraph G_s^h returned by TBFH-1.Considering Property 1, TBFH-2 is able to compute paths using internal edges and so guarantees the computation of optimal 1-alternate paths.

The set A returned by TBFH-2 contains as many couples of alternatives as the number of destinations if the network is 2-edge connected. This set of candidate next hops serves as a basis to locally verify the validity of each alternate path. Lines 8-11 performs the virtual links updates while lines 12-21 performs the remainder of the SPF algorithm in each reduced subgraph. The boundaries of each subgraph G_s^h are here implicitly marked thanks to the condition $y \in N^h$ in line 15.

Fig.3 illustrates the basics of the TBFH computation. Note that we use the same caption as in Fig. 2. Moreover, for the sake of clarity, we do not have represented all potential internal links and links belonging to branches in this example. In Fig.3(a), TBFH-1 is illustrated on a simple graph with three branches attached to root s: nodes are gathered depending on their respective primary first hop (a, b and c nodes) to form branches. TBFH-1 takes advantage of simple alternate paths by adding them as *virtual links* on each subgraph. Each virtual link is included as a directed edge in its associated subgraph $G_s^h(N_s^h, E_s^h, w_s^h),\ h \in a, b, c$. For example, the optimal simple alternate path $P_1(s, u) \circ (u, z)$, which uses the transverse edge (u, z) linking $branch_b(s)$ and $branch_a(s)$, is added as a virtual link in $G_s^a(N_s^a, E_s^a, w_s^a)$ with a weight of $C_1(s, u) + w(u, z)$ because z belongs to $branch_a(s)$.

In Fig.3(b), we have illustrated TBFH-2 on $branch_a(s)$. The two virtual links allowing to reach $branch_a(s)$ are depicted as dashed arcs. The subgraph $G_s^a(N_s^a, E_s^a, w_s^a)$ contains all virtual links connecting s and nodes in $branch_a(s)$ having an incoming transverse edge (y and z on the example). The first hop (s, a), between the root node s and $branch_a(s)$ is not added to the subgraph G_s^a in order to only focus on alternate paths computation. Therefore, the SPF computation on G_s^a allows s to compute the shortest path towards each destination $d \in N_s^a$ in this subgraph. A shortest path in a subgraph $G_s^h(N_s^h, E_s^h, w_s^h)$ corresponds to the shortest path in $G \backslash (s, h)$. In other terms, the optimal 1-alternate path in G such that the cost $C_2(s, d),\ \forall s, d \in N$ computed in a graph G is equal to the cost $C_1(s, d)$ in the graph $G \backslash (s, NH_1(s, d))$.

Thus, we have the following property.

Property 2: TBFH allows any node $s \in N$ to compute its

set of optimal 1-alternate path towards any destination $d \in N$.

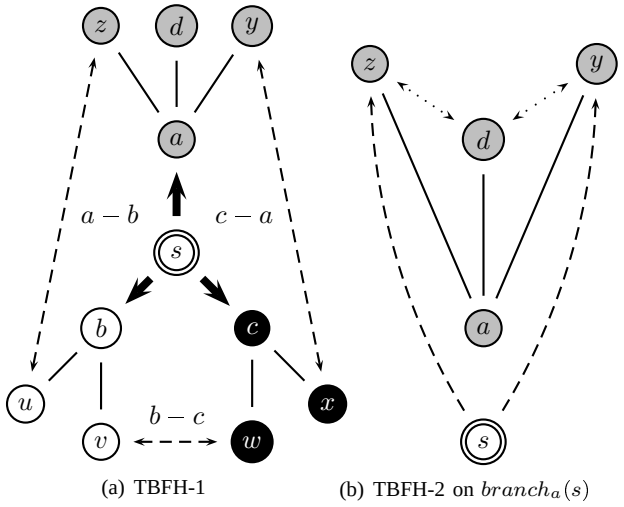
Proof: Let us consider that there exists a 1-alternate path $\mathcal{P}$ whose cost is lower than the one computed by TBFH $\mathcal{P}'$: $\Rightarrow C(\mathcal{P}') \geqslant C(\mathcal{P})$. Considering a given pair (s,d), let us denote such a path $\mathcal{P}(s,d) = p_1 \circ (x,y) \circ p_2$ where d belongs to a given $branch_h(s)$, $h \in NH_1(s)$. Since $\mathcal{P}(s,d)$ is an optimal 1-alternate path as defined in section III-C, p_1 is a primary path, (x,y) is a transverse edge and p_2 a path not containing any transverse edge. If $p_2 = \epsilon$ ($d = y$) then $\mathcal{P}(s,d)$ is a simple alternate path.

First, let us consider that the optimal simple alternate path $p_1 \circ (b,c)$ is not computed with TBFH. We know that when the minimum key node extraction returns a given node, then all its outgoing links are explored in the following iteration step. Therefore, TBFH explores all the composition of a primary path with a one hop path containing a transverse edge. By definition, this composition forms a simple alternate path, and thus TBFH computes all optimal simple alternate paths.

Then, the only subpath of $P(s,d)$ that can differ from the 1-alternate path computed with TBFH is p_2. Node y is the first hop of the path p_2. Thanks to the property of optimality of sub-paths, we know that, if the best alternate path towards d goes through y, TBFH-2 is able to compute the optimal cost between y and d not containing any edge in $trans(G,s)$. If we apply the Dijkstra algorithm to the graph $G(N, E', w) \mid E' = E \backslash \{trans(G,s), (s,h)\} \bigcup \{Pt'(s,d)\},\ \forall h \in NH_1(s),\ \forall d \in N$ where $Pt'(s,d)$ denotes the set of virtual links representing simple alternate paths $\in Pt(s,d)$, it is equivalent to applying it on each subgraph $G_h^s,\ \forall h \in NH_1(s)$. Indeed, the removal of s partitions $G(N,E',w)$ into $|NH_1(s)|$ connected subgraphs. If y is not included in one of the alternate paths towards d computed by TBFH, then it means that $\mathcal{P}'(s,d)$ has a cost strictly lower than the one of $\mathcal{P}$: $\Rightarrow C(\mathcal{P}') < C(\mathcal{P})$. Thus, we have the following results: $C(\mathcal{P}') \geqslant C(\mathcal{P})$ and $C(\mathcal{P}') < C(\mathcal{P})$, which is impossible. ■

Thus, considering Property 1, TBFH allows any node $s \in N$ to compute its set of local post convergence next hops for all destination $d \in N$. TBFH-1 computes all optimal simple alternate paths. TBFH-2 computes optimal paths on each subgraph $G_s^h,\ h \in NH_1(s)$. When TBFH-2 applies a SPF algorithm on a subgraph G_s^h, it uses virtual links representing optimal simple alternate paths rather than using the first hop (s,h). Since these paths are optimal alternates not using the first hop (s,h), we can deduce that there cannot exist an alternate path whose cost is strictly lower than those that TBFH-2 computes. However, note that there may exist a valid loop-free next hop which is not a local post convergence one.

Fig.3 also helps to understand the cases where an optimal 1-alternate candidate is not valid according to a given loop-free rule whereas there exists a valid k-alternate path ($k > 1$). Let us consider the DC rule, and assume there exists no valid 1-alternate path linking s and d. In particular, let us assume that $c = NH_2(s,d)$ is not a valid next hop because $C_2(s,d) - w(s,c) \geqslant C_1(s,d)$. Let us assume that the third ranked next hop, $NH_3(s,d) = b$, is the first hop of a 2-alternate path $\mathcal{P}_3(s,d) = (s,b) \circ P_1(b,v) \circ (v,w) \circ P_1(w,x) \circ (x,y) \circ P_1(y,d)$



(a) TBFH-1 (b) TBFH-2 on $branch_a(s)$

Fig. 3. TBFH algorithm illustration

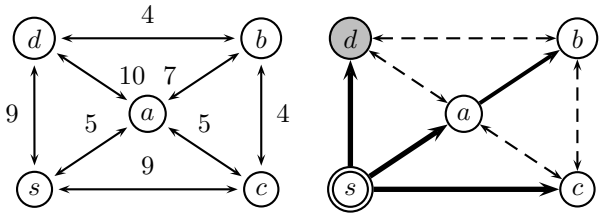
because the weight of (u,z) is large. If b is a valid next hop such that $C_3(s,d) - w(s,b) < C_1(s,d)$ whereas c is not a valid next hop, we know that:

$$C_1(b,v) + w(v,w) + C_1(w,x) < C_1(c,x) \tag{5}$$

This inequality illustrates the kind of cases which are problematic: TBFH does not always compute a valid alternate next hop while there exists one. However, in practice, problematic cases do not occur frequently as shown in Sec. VI-C. On the one hand, if $c = x$ or if $c = w$, a k-alternate path cannot be valid if there does not exist a valid 1-alternate path. More generally, the shorter the distance between c and x or c and w is, the more this case becomes rare. On the other hand, we also know that $w(s,c) + C_1(c,x) \leqslant w(s,b) + C_1(b,v) + w(v,w) + C_1(w,x)$, so it means that $w(s,b) > w(s,c)$. Therefore, this singular case may occur when the distribution of local link weights is very heterogeneous. Note that if the IGP weights of all links attached to a given router are the same, then TBFH computes the complete DC coverage for this router. If the link valuation is globally uniform, then TBFH computes the complete set of valid DC and LFA next hops towards any destination.

In practice, there can exist much more than one transverse edge linking the same pair of branches. If there exists several transverse edges between $branch_b(s)$ and $branch_c(s)$, the left part of the inequality becomes the minimum value between all possible 1-alternate path combinations linking $branch_b(s)$ and $branch_c(s)$.

Fig. 4 gives a basic example of the TBFH computation on a valued topology. Weights are given in Fig. 4(a) and we assume that they are the same in both directions of each link. Fig. 4(b) provides the partition of edges according to our terminology with s as root node. We can notice that there are three subgraphs rooted at nodes d, a and c. Using TBFH, s is able to compute all DC next hops except one: for the pair (s,d), there exists a 2-alternate path (s,c,b,d) which is not computed by TBFH whereas the cost of the path (c,b,d) is



(a) The weighted topology (b) The branch decomposition

Fig. 4. TBFH advantages and drawbacks

strictly lower than the primary one between s and d ($4+4 < 9$). This drawback is due to the fact that TBFH computes only the post convergence path (s, a, d) whose cost is lower than the one of (s, c, b, d) ($5 + 10 < 9 + 4 + 4$). This case illustrates the inequality (5) where: $w(c, b) < C_1(a, b) = w(a, b)$. Note that other terms do not appear because this example does not involve several path segments but only the transverse edge (c, b).

E. Complexities

In this section, we analyze the time and the space complexity of the TBFH algorithm. A SPF algorithm such as TBFH or Dijkstra's algorithm uses a structure called a priority queue (PQ) to perform its computation[2]. This structure is used to store and evaluate the costs of the explored paths and supports three methods: *extract_min*, *decrease_key* and *insert_key*. The first operation allows for finding the current minimal cost path in the PQ, while the two other operations respectively modifies and creates a new path cost entry in the PQ. The time complexity of a SPF algorithm depends on the kind of PQ used. Let us denote m the cost of the operation *extract_min*, d the cost of the operation *decrease_key*, and i the cost of the operation *insert_key*. The operation *extract_min* denotes both the search of the minimum key and its suppression. The complexity of a SPF algorithm (e.g, the Dijkstra SPF) is then:

$$O((m + i) \times |N| + d \times |E|)$$

With an array list as PQ, $m = O(|N|)$, $i = d = O(1)$ whereas a binary heap reduces the cost of m to $O(log_2(|N|))$ while the cost of i and d are increased to $O(log_2(|N|))$. The optimal structure for a SPF PQ is the Fibonacci heap [12]. The amortized cost of m is then in $O(log_2(|N|))$ while $d = i = O(1)$ on average. With a Fibonacci heap PQ, the amortized time complexity of TBFH, is lower than:

$$2 \times O(|N|log_2|N| + |E|)$$

Proof of the TBFH time complexity: Let us denote $\{|N_1|, |E_1|\}, ..., \{|N_k|, |E_k|\}$ the size of each branch rooted at a given root s in terms of node and edge number ($k \leqslant deg(s)$). Then the complexity of TBFH is $SPF(|N|, |E|) + SPF(|N_1|, |E_1|) + ... + SPF(|N_k|, |E_k|)$ where $SPF(a, b)$ denotes the complexity of a SPF algorithm on a graph containing a nodes and b edges. Indeed, the term $SPF(|N|, |E|)$ is the time complexity of TBFH-1 whereas $SPF(|N_1|, |E_1|) + ... + SPF(|N_k|, |E_k|)$ is the time complexity of TBFH-2. We know that $SPF(|N_1|, |E_1|) + ... + SPF(|N_k|, |E_k|) \leqslant SPF(|N|, |E|)$. Indeed, we have $SPF(|N|, |E|) = O((m + i) \times |N| + d \times |E|)$ and we know that $|N_1| + ... + |N_k| = N - 1$ and $|E_1| + ... + |E_k| \leqslant |E| - k$. Yet, even with a Fibonnaci heap to model the PQ, the function $SPF(a, b)$ is supra-linear with a logarithmic factor: $SPF(a, b) = alog(a) + b$. ■

In a favorable case, if the branches are well balanced such that $\forall h \in NH_1(s),\ |N_s^h| \sim \frac{|N|}{|NH_1(s)|}$, then the overall run time complexity of TBFH is in:

$$|N|log_2(\frac{|N|^2}{|NH_1(s)|}) + 2 \times |E|$$

Note that the time complexity of TBFH is minimal when there is a large number of branches well balanced in terms of number of nodes. Indeed, the additionnal time complexity induced by TBFH-2 is dominated by the term: $p\ log_2(p)$ where $p = max(|N_s^i|),\ \forall i \in NH_1(s)$, is the number of nodes in the largest branch.

If $deg(s) = 2$, TBFH provides an optimal coverage (the same as kSPF) with a time complexity lower than two SPF computations. When $deg(s) \geqslant 2$ and if IGP weights are not equals, TBFH does not guarantee the computation of the complete set of loop-free alternate next hops. However, TBFH guarantees the post convergence property and a controlled time complexity overhead while it provides an almost complete coverage as shown in Sec. IV-B and VI-C. Note that for the LFA validation, it is necessary to compute an additional reverse SPT rooted at the calculating node (Sec. II-B).

The additional space complexity required by TBFH only depends on the number of 1-alternate paths computed per destination. With the basic version of TBFH, this additional space complexity is then only $2 \times |N|$ (compared to a single SPT computation), because at most, TBFH only needs to store two candidate next hops per destination at any time during the computation.

IV. Deployment Discussions

In this section, we consider practical issues related to the deployment of TBFH. First of all, note that our proposal is incrementaly deployable such that routers implementing TBFH can coexist with non multipath routers or ECMP routers. Indeed, using TBFH, the DC and LFA rules can be verified locally without exchanging any messages. The global loop-freeness property is still guaranteed.

We are interested by two practical issues: how to improve the coverage provided by TBFH without strongly increasing its time complexity ? How to deal with TCP ?

A. Coverage and Complexity Tradeoff

It is possible to increase the coverage of TBFH by adding a time and space complexity of $deg(s) \times |N|$. Instead of a time complexity of $deg(s) \times SPF$ using kSPF, we propose a TBFH improvement (TBFH') requiring a time complexity of $2\ SPF + deg(s) \times |N|$. For that purpose, it is necessary

[2]Note that the PQ is denoted Tc in Alg. 1

to use a matrix containing a best cost per destination per next hop. Indeed, there may exist several simple alternate paths towards a given destination. In the same way that it is necessary to tune the basic SPF algorithm to compute all equal cost multiple paths, TBFH can perform multiple inheritances. Towards a given destination, TBFH-1 needs to take all optimal simple alternate paths using different first hops into account[3] before proceeding to the second phase. TBFH-2 can use those multiple paths and costs to increase the number of valid 1-alternate paths. Note that using an additional time complexity of $deg(s) \times |E|$, we can still increase the coverage by considering a subset of k-alternate paths ($k \geqslant 1$) such as described in [30]. This improvment is denoted TBFH" in Sec. VI. In particular, TBFH" allows to compute all k-alternate paths having a cost equal to the one of the primary path (in the same way that it is necessary to improve SPF algorithms in order to compute ECMP next hops).

In order to compute a complete set of valid alternate next hops towards any destination with any link valuation, it is possible to use a variant of TBFH: Two Valid First Hops (TVFH). According to the objective, TVFH needs to take a modified link valuation into account. Let us denote w' the new valuation function. If the goal is to find loop-free next hops for load balancing purposes using the DC rule, the function w' must return the same weight for all outgoing links. Using a valuation function w' such that:

$$w'(x,y) = \begin{cases} \lambda & \text{if } x = s \ (\lambda \in \mathbf{R}) \\ w(x,y) & \text{otherwise.} \end{cases}$$

TVFH computes a complete set of DC next hops with only one additional SPF computation compared to TBFH.

The function w' can also be tuned to compute LFA next hops. If w' verifies:

$$w'(x,y) = \begin{cases} \lambda - C_1(y,x) & \text{if } x = s \ (\lambda \in \mathbf{R}) \\ w(x,y) & \text{otherwise.} \end{cases}$$

Therefore, TVFH computes an optimal LFA coverage although the time complexity does not depend on the degree of s. In this case, we need to perform two additional SPF computations compared to TBFH because we also need to compute the *reverse costs* $C_1(y,x), \ \forall y \in succ(x)$.

Note that with TVFH, the optimality of the computed valid alternate paths is not guaranteed. There may exist a shorter valid alternate path because weights of each outgoing link of the calculating node have been modified. For instance, with the DC rule, the w' valuation function considers the best alternate path from the neighboring node, not from the root node. To perform a good tradeoff between alternate path optimality and coverage, we may consider the union of the ouput of TBFH and TVFH. For a given pair (s,d), if there exists a valid local post convergence next hop, s computes it using TBFH, otherwise, if there exists a valid next hop towards d, s can compute it using TVFH.

Another possible extension of our proposal is to use TBFH to compute and validate remote exit points for fast reroute purposes [8]. This technique also refers to tunnels and directed tunnels described in [18]. A remote exit point $x \in N$ associated to a given neighbor $v = NH_j(s,d)$ verifies:

$$C_j(s,d) - C_1(s,x) - C_1(x,s) < C_1(s,d) \quad (6)$$

Note that we have $v = NH_j(s,d) = NH_1(s,x)$: the best next hop used towards x is the same than the one used for verifying this rule. Indeed, in addition to the fact that the elementary path $P_j(s,d)$ must go through x and be the optimal one via the edge (s,v), we want that the deflected encapsulated trafic between s and x uses the shortest path between s and x. Rule (6) is a rewriting of rule (4) for remote exit point nodes. Using TBFH, it is possible to consider the penultimate next hop of optimal simple alternate paths as candidate exit points.

To achieve a complete coverage, we need an additional step if there is no candidate exit points verifying rule (6). Indeed, if IGP weights are symetric and the network is 2-edge connected, TBFH is then able to provide a full coverage for any single link failure thanks to an IP-in-IP tunneling mechanism. If a router s detects a link failure impacting a set of destination $\{d\}$ which are not covered, it has to (pre)compute a directed tunnel to allow packets destinated to the set $\{d\}$ to circumvent the failure by encapsulating them towards x.

If we consider the set of optimal 1-alternate paths as directed IP-in-IP tunnels, we can easily prove by construction that the rerouted packets will never loop. Let us consider the triplet (s, d, $NH_1(s,d)$), and the optimal 1-alternate path $P_1(s,x) \circ (x,y) \circ P_1(x,d)$ where (x,y) is the unique transverse edge of the path. To avoid the failed link $l = (s, NH_1(s,d))$ impacting this triplet without creating a forwarding loop, a router s can follow this simple guideline: when l fails, rerouted packets destinated to d are encapsulated towards the penultimate hop of the optimal simple alternate path $P_1(s,x) \circ (x,y)$ and then *pushed*[4] to the final hop (the head of the transverse edge: y). Thus, rerouted packets are forwarded through the shortest path between s and x, $P_1(s,x)$, pushed through the transverse edge (x,y) and finally forwarded through the primary path $P_1(x,d)$. The loop-free property is guaranteed if $C_1(y,d) < C_1(y,s) + C_1(s,d)$. This inequality is ensured by the design of simple alternate paths and the symetry of IGP weights because $C_1(y,d) \leqslant C_1(y, NH_1(s,d)) + C_1(NH_1(s,d),d) < C_1(y,s) + C_1(s,d)$.

Let us consider the topology given in Fig. 3(a), and the triplet (s, d, $a = NH_1(s,d)$). If there do not exist any LFA next hop covering the link (s,a) for the traffic towards d, s can encapsulate the traffic destinated to d towards x and forces x to push the deflected traffic through its direct link (x,y). Note that if the optimal 1-alternate path goes through $branch_c(s)$ and $C_2(s,d) - C_1(s,x) - C_1(x,s) < C_1(s,d)$, then the tunnel towards x is sufficient: x does not need to push the traffic through y, it will anyway use y as its first hop towards d. To avoid the per destination basis of such an approach, we can also consider the set of destinations

[3]Not only the best alternate first hop as it is the case with Alg. 1

[4]The term pushed refers to a secondary encapsulation if x does not use y to forward its traffic towards d

belonging to $branch_a(s)$, and use the shortest simple alternate path towards this branch as the directed tunnel to protect the link (s, a) without considering the optimality of emergency routes at the granularity of the destination.

B. Multipath Routing and TCP

One of the main issues of a multipath routing architecture is its interaction with TCP. This interaction can lead to the head of line blocking problem ([22]). In practice, if the characteristics of the multiple paths connecting a given pair of hosts are heterogeneous (in terms of latency, bandwidth, ...), packets following those paths can reach the destination in an order differing from the sending sequence. With TCP, out-of-sequence segments have to be immediatly acknowledged, and the sender receiving multiple duplicated acks triggers congestion mechanisms resulting in the reduction of the congestion window. To avoid this issue, packets belonging to a given flow must be forwarded through the same path. We call such a rule the *path consistency rule*. The use of a path selector field in the IP header is a convenient solution. Each packet of a given flow is tagged, and this tag is used by routers to ensure the path consistency rule.Thus, the multiplexing of paths is statistically ensured by the huge number of flows in the network. Note that it is possible to use a different path selector between each TCP burst (packets in the same sender windows). In practice [21], a mechanism such as ECMP applies a CRC-32 hash function on some fields of the IP header to emulate the use of a dedicated path selector.

To improve ressource pooling and thus increase the throughput of a given application, multipath transport protocols can use several spaces of packets sequencing at two levels (at the sub-flow and at the meta connection levels) to define several sub-flows for a given pair sender/receiver (e.g, [22] or [39]). Each sub-flow is then associated with a given tag corresponding to a given path. However, the problem remains the same at another level: the meta-buffer of the receiver must deal with the head of line blocking issue. Here the term meta buffer refers to the buffer used to gather the data of each different sub-flows before passing it to the application layer. Thus, either the set of paths has to be homogeneous enough in terms of capacity/latency/loss, or the meta-buffer has to be large enough to limit this issue. This is an open issue mentioned in [24]. However, the path selector only solves the preservation of each sub-flow consistency.

Failures introduce another level of complexity. Indeed, when a router detects a failure, the routing protocol must converge to a new stable forwarding situation, e.g., a new association [destination, primary next hop]. Then, a given flow whose forwarding path went through the failed component is deflected to a new path even with the use of a path selector. The side effects of such an issue are partially described in [21] with the ECMP protocol and are not negligible. With fast re-routing techniques such as LFA, some flows can be deflected twice. Considering one of these flows[5], if the LFA next hop is not a local post convergence one, the flow is first deflected via the LFA next hop and then, once the routing protocol has converged, to the post convergence next hop. We argue that the side effect of using non post convergence LFA next hops can introduce more inconveniences than advantages. Indeed, the TCP head of line blocking issue is exacerbated by multiple path changes (flapping), reducing the advantages of fast re-routing schemes [6].

We envision a scheme minimizing the flapping of a given flow and allowing a good tradeoff between path exploration by the end hosts and forwarding simplicity in the network core. However, note that our path computation algorithms also allow to perform a router based load balancing inside the network core. We are interested in the following architecture: the network proposes several routes while the end hosts use a tag to select their forwarding path. In practice, flows are tagged with a path selector, and routers apply a given function on the path selector to assign the flow to a given next hop. Note that the forwarding function should use the tag differently on each router (e.g., uses different subset of bits as the key).

There are several examples of tagging methods (e.g, [46] and [32]) allowing to explore the path diversity provided by the network. The main idea behind this kind of schemes can be summarized this way: for the sake of clarity and without loss of generality, let us assume that each router has two valid next hops towards every destination and that the path selector is encoded on N bits. If each router uses one bit of the tag to select the forwarding next hop, it allows an end host to explore 2^N routes if all routes have at least N hops. For that purpose, each router needs to know the bit (or the subset of bits to generalize) that it must read to perform its next hop selection. There are several ways to enable this kind of technique: use another dedicated field, shift the last bit used, etc.; but generally this kind of solution implies an IP header modification at each hop (where there exists several next hops to a given destination). To avoid this issue, it is possible to use a local identifier (or a modulo operation of an IP header field such as the TTL which changes at each hop) to select another combination of bits belonging to the path selector (the order of projection must not be permutable between each hop). Typically, the path selector field can be included in the flow label of an IPv6 header or using TCP ports for IPv4/IPv6. It is possible to use the layer-4 transport ports if there is no NAT or middle-boxes between the two hosts.

To perform an optimal path exploration (i.e, all possible forwarding paths are explored), the path selector should not be used identically by each router (or equivalently), otherwise

[5]Such a flow continues to use a route including the node with the failing link

[6]Note that the re-computation of new sharing ratio after a failure also exacerbates the head of line blocking problem. Let us consider several next hops and a proportional static sharing ratio towards a given destination: if one of this next hops fails the proportions have to be re-computed and even flows not directly impacted by the failures can be redirected via a different next hop.

the number of paths potentially explored can become low [7]. Considering a pair (source,destination) and the set of routes R linking these two routers, the path exploration is optimal if, for any routes $z \in R$, each router $r \in z$ can use a number of bits n verifying $2^n \geqslant m$ where m is the number of valid next hops for the pair (r, destination). Note that each router on a given route z should use a different subset of bits. Thus, the size (in number of bits) of the path selector must be greater than $\lceil log_2|R| \rceil$ where $|R|$ is the total number of routes. Using an arbitrary number of bits N in order to limit the size of the path selector, it is possible to explore 2^N routes if there exists at least N routers on each route $z \in R$ having at least two next hops towards the destination.

Our goal is to deploy such an approach without forgetting to take into account the side effects of failures in order to improve the efficiency of LFAs. To minimize the flapping due to failures, we propose a fast re-routing scheme selecting LFA next hops which are also local post-convergence next hops. Our approach focuses on the set of next hops used after the convergence of the routing plane. Then, it locally[8] minimizes the flapping to at most one change per flow per failure instead of two. We have shown in Sec. VI-C that LFA next hops are generally post convergence next hops: by removing LFA next hops that are not post-convergence, we loose less than 7% of coverage (between 1% and 7% on real weighted topologies). Our algorithm guarantees two properties:

(i) the post convergence nature of LFA next hops, e.g., we minimize the head of line blocking issue.
(ii) a computation time inferior to two SPF computation instead of a computation time of deg(s) SPF.

We believe that those two advantages are good incentives to use our approach. In our combined approach (load balancing and fast re-routing modes), note that we can also validate non post convergence LFA next hops which verify the DC rule considering the optimal alternate path cost instead of the primary one. Indeed, it allows to increase the coverage without loosing the post convergence property (i) because those next hops will be used as DC next hops for load balancing after the convergence of the routing plane. To summarize the general approach, let us consider that each source tags its packets with a given path selector. When such a packet arrives on a router r, r applies a specific hash function on the path selector to compute the next hop to use. The projection ratios of the hash function have to be re-computed each time a topology change occurs to remove failed next hop (or add new ones). To limit those changes in case of failures, we focus on MP and FRR that are post convergence next hops. In addition, we argue that load sharing ratios re-computation for the PR/MP set is not necessary during the transient period of topology changes, such that we simply deflect impacted packets to another viable path until the new FIB has been updated.

[7]Note that if each consecutive routers uses a different hash function projecting the subset of path selector combinations it receives towards a large subset of its valid next hops, the path exploration can be close to the optimum without IP header modification. In a same way, using a local identifier to modify the behavior of a global hash function, it is possible to statistically achieve a near optimal path exploration: the necessary condition is to perform successive hashes perturbing the order of the path selector projection.

[8]Note that some flows can be deflected uptream of the failure after the convergence of the routing plane. Then, those flows will be deflected twice anyway.

V. Related work

Compared to the basic kSPF algorithm, another way to compute a set of multiple paths is to use an enhanced Shortest Path First algorithm to locally compute multiple paths for each destination. For example, algorithms and implementations presented in [36] have been designed to compute the set of K-shortest elementary paths. However, those algorithms do not guarantee that computed paths are first hop distinct. The K-shortest paths problem is not suited for hop by hop forwarding. Indeed, in order to forward packets via these K explicit paths, a signaling protocol is necessary to establish the forwarding paths from the ingress router towards each egress router.

Eppstein [16] also proposes several algorithms for the K-shortest paths problem, and presents in [15] an almost exhaustive bibliography of this issue. However, most of these solutions are not applicable for hop by hop multipath routing since the computed paths are neither elementary nor first hop disjoint. Note that Suurballe [40] proposes an algorithm for disjoint paths computation. This issue is more restrictive than the computation of first hop disjoint paths and so reduces the number of candidate next hops.

Another related work enhancing such a SPF computation is a proposition of Topkis [42]. He presents an algorithm computing the K-best first hop disjoint alternate paths whose time complexity is in $O(K\ SPF)$. If the ranks of valid alternate next hops, as defined in Table I, are lower than the degree of the router, this algorithm allows to reduce the time complexity required for the validation phase (compared to kSPF). However, in the worst case, if all alternate paths have to be computed to ensure the complete termination of the validation phase, then this algorithm induces the same complexity as kSPF. Furthermore, TBFH allows to uncouple the primary next hops set computation from the alternate next hops validation into two consecutive phases.

The use of incremental SPF methods (e.g, [28] or [34]) allows to quickly re-compute a new SPT using the knowledge of the previous computed one. The required time complexity depends on the position of the topology change in the SPT. If the change is local to the calculating router, the gain in time complexity can be unsignificant. Note that Cisco routers use a full SPF to handle local failures [2].

Chen and al. propose in [10] a multipath method based on path suffix similar to the work presented in [26]. In this case, the forwarding mechanism changes the destination lookup paradigm. The forwarding scheme depends on extra information carried in each packet: a path suffix field which indicates the remainder of the path to use in order to prevent the formation of forwarding loops. This technique induces a change in the forwarding plane of all routers whereas our approach is incrementally deployable.

In [46], Yang and Wetherall introduce a set of loop-free rules whose flexibility allows to increase the number of valid neighbors. This set of rules implies that the forwarding mechanism has to be specific to the incoming interface. It allows forwarding loops at the router level but not at the link level: a packet is never forwarded through the same link twice but can enter the same router twice. Thus, delays can increase if paths contain several times the same router: it may unnecessarily consume more resources (e.g, routers CPU, bandwidth).

In [35], authors present a multipath computation algorithm called MARA calculating a directed acyclic graph per destination. This approach allows to generate a large set of forwarding paths optimizing several objectives such as the maximum connectivity. However, the time complexity of their approach is in $O(|N| \times SPF)$.

VI. Evaluation and Implementation

A. Methodology

We use our own SPF computation framework (available online at [3] and coded in C for efficiency) to compare several routing approaches. We have implemented a classical Dijkstra SPF and extended it to support TBFH and kSPF algorithms. We have also implemented the DC and LFA rules for the path validation phase. Using a large variety of topologies, we show that our algorithm is able to compute almost the same path diversity as kSPF but with a lower time complexity as demonstrated in Sec. VI-B.

We present results obtained on three different kinds of topologies: actual, inferred and generated topologies. The first category of networks are real topologies with their IGP weights (for confidentiality, we anonymize some of them and we approximate the dimensions given in Table IV). Topologies denoted ISP1 and ISP2 are commercial networks covering an European country. ISP3, ISP4, ISP5 and ISP6 are Tier-1 ISP networks. The second category of topologies were chosen among the Rocketfuel inferred set of networks given in [27]. The inferred IGP weights are also provided for the subset of topologies that we select.

Finally, we have also used the IGEN topology generator [37] in order to obtain a set of homogeneous topologies of various sizes. We have generated a set of topologies containing between 500 and 5000 nodes using the *Delaunay-triangulation* heuristic (see [38] for details). The Delaunay-triangulation parameter offers a great degree of path diversity compared to other heuritistics provided by IGEN. Networks providing a high degree of path diversity stresses the SPF computation: this is useful to evaluate SPF algorithms performances. The IGP weights used for those topologies are either fixed to 1 (denoted f in Fig. 7) or uniformly distributed with integer values belonging to the range $[1, 10]$ (denoted u in Fig. 7).

We present two kinds of results. The first highlights the low time complexity of TBFH while the second shows that our algorithm provides a coverage close to the optimum (using kSPF).

B. Time Complexity Results

In this section we analyse the time complexity of TBFH compared to kSPF. For that purpose, we perform realtime measurements. Note that our computation time results are absolute (e.g., they depend on the CPU we use) and given in micro seconds, but the relative comparison between the different algorithms we consider is meaningful. Mesurements have been performed on a CPU with a frequency of 2.4Ghz. Furthermore, note that we have implemented an improved version of the basic kSPF in order to offer a more competitive comparison. kSPF denotes here an improvement based on a computation entirely rooted at the calculating node. Indeed, it is possible to reduce the computation time considering the nature of DC and LFA loop-free rules: a neighbor using the calculating node as its primary first hop for a subset of destinations cannot be a DC or a LFA next hop for those destinations. In order to avoid the computation of useless candidate next hops, we perform a specific operation per neighbor. Our improvement of kSPF needs as many SPF computations as the basic kSPF, but by removing all edges connected to the root node except the one towards a given neighbor at each iteration, we are able to optimize the computation. Indeed, if a given neighbor is unable to reach a large subset Z of destination nodes without going through the calculating node, then the SPF time complexity for this neighbor depends on $|N| - |Z|$ (and not on $|N|$). Thus, if the network is not 2-node connected (the removal of the calculating node partitions the graph in several connected components), then this kSPF optimization can save many CPU cycles.

To optimize a SPF computation, the first important choice is the kind of PQ. We have implemented four kinds of PQ. The first one (denoted *AL*) uses a *naive* static array list, while the second one (denoted *LL*) uses a doubly linked list to reduce the cost of the *extract_min* operation. The two other PQ are more sophisticated: our third PQ (denoted *LL+*) also uses a doubly linked list but optimizes the *extract_min* operation whereas our last PQ (denoted *B-HEAP*) uses a binary heap structure. Note that the LL+ PQ provides an amortized time complexity of $O(N)$ per SPT computation if the link valuation is the same for all links (as with a B-HEAP PQ). More details about various PQ implementations can be found in [47].

The LL+ PQ takes advantage from the specific primary path cost distribution in IP networks. Indeed, when shortest path costs are distributed in a way that favor the probability of extracting successively a same minimal cost during the SPF execution, the time complexity of a SPF algorithm can be strongly reduced. In practice, if there exists a low number of shortest path costs compared to the number of nodes, LL+ allows to achieve an amortized linear time complexity. In a simple case, when IGP weights are equal, the number of different shortest path costs is equal to the network diameter which is generally very low compared to the number of nodes. In usual IP networks, IGP weights are distributed in a small subset of values corresponding to link capacities. Thus, the additive combinations of such weights are also limited.

This observation favors the use of LL+ in most of actual networks. Obviously, if IGP weights are chosen among a large set of floating values (such as with Abilene which uses link latencies instead of link capacities), LL+ does not result in any performance gain.

Technically, LL+ is an extension of a LL PQ where the PQ structure is identical: a doubly linked list. However, the operation *extract_min* is slightly modified:

- If the head of the linked list has a cost equal to the current minimal one (the cost towards the last explored node), the *extract_min* operation directly extracts the head without searching for the min: there cannot exist a cost strictly lower than this one.
- Otherwise (the head key cost is greater than the current min), the *extract_min* operation must perform its goal: searching for the min. During this phase, all costs equal to the last min founded are consecutively linked to the head of the doubly linked list.

Moreover, note that the *insert_key* operation must insert new explored keys to the tail of the list (its cost is necessarily strictly greater than the current min). Using LL+, the number of min searching is strongly reduced compared to LL. Indeed, if Z denotes the number of different primary path costs, then LL+ only performs Z min searches. Hence, the time complexity of a SPF using LL+ is in $0(ZN + E + N - Z)$. When Z is low which is typically the case on actual IP networks, the time complexity provided by LL+ is near-optimal. If the metric is the number of hops, then Z is equal to the network diameter.

For the sake of clarity, we only use the PQ providing the best performances on average (for realistic topologies, see 5 and Table IV): LL+. SPF performance comparisons are not realistic with a non efficient PQ. Using an efficient PQ allows us to compute a lower bound on the time complexity gain provided by TBFH. With a naive PQ, we may overestimate the performance of TBFH compared to kSPF because the use of non efficient PQ can hide the cost of additional optimization operations. The gain provided with an optimized PQ is generally asymptotic: it is only achievable for sufficiently large IP networks. Fig. 5 illustrates this behavior (note that the y-scale is logarithmic). For small actual networks, optimized structures such as a B-HEAP degrades the performance compared to LL+ (and even compared to a simple LL for very small networks whose nodes number is lower than 50). However, for large actual networks (ISP6 on Fig. 5, the gain is asymptotic: the larger is the network, the higher the gain provided by the B-HEAP is. In particular, this gain is achieved when the topology is highly meshed because the number of keys contained in the PQ can be large during the SPF computation. For large generated topologies (shown in Fig. 6 with the u-parameter), the B-HEAP does not provide any gains because the PQ never contains a sufficient number of keys. Indeed, the main gain of heaps comes from the log factor applied to the number of keys contained in the PQ. Consequently, we do not have implemented a Fibonnaci Heap because its gain is even more asymptotic than a B-HEAP while usual IP networks are rarely larger than 1500 nodes.

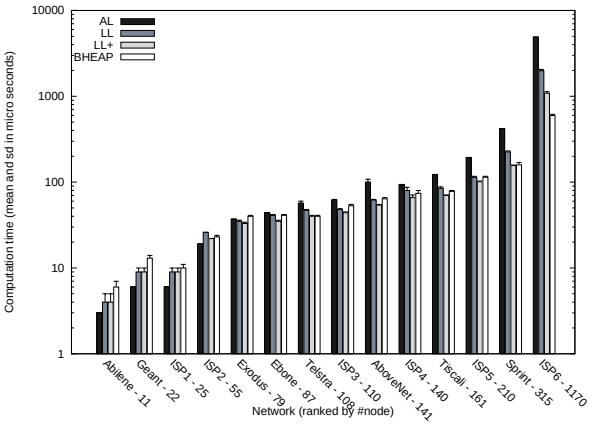


Fig. 5. PQ performance comparison on actual and inferred networks

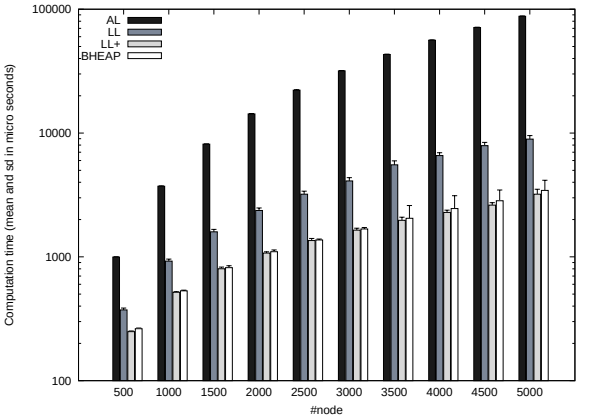


Fig. 6. PQ performance comparison on generated networks

Using our LL+ PQ implementation, we evaluate the performance of TBFH compared to a kSPF algorithm. Considering all nodes in the generated network, Fig. 7 plots the average computation time required by kSPF and TBFH to compute their sets of primary and candidate next hops. We firstly notice that TBFH allows to save many CPU cycles: for large topologies, the average computation time is almost divided by two with TBFH compared to kSPF. We also observe that the cost of SPT computations are greater for topologies with non fixed weights: the computation time becomes greater with the u-parameter. This loss of efficiency is due to the design of the PQ: the number of different primary path costs is higher with topologies using heterogeneous IGP weights.

Then, we can emphasize nodes where the CPU gain is the most important. Table III gives a closer look at the computational time for all our actual networks (their dimensions are given in each figure caption: $(|N|, |E|)$). We plot the average computation time required for ECMP, kSPF and TBFH according to the node degrees. Note that we also

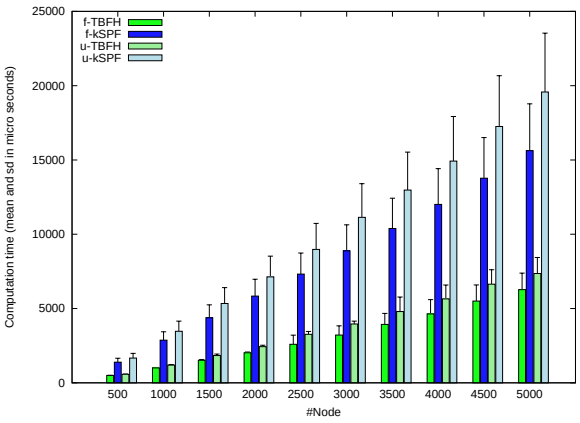


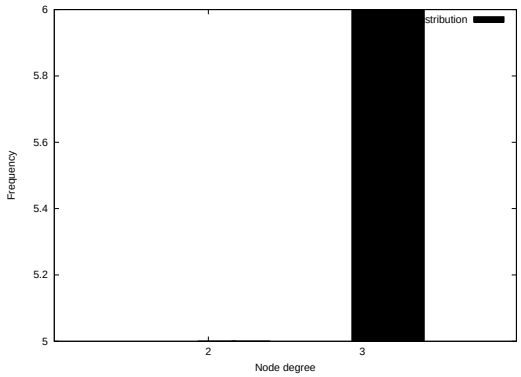
Fig. 7. Computational time of kSPF vs. TBFH (IGEN networks)

provide node degree distributions.

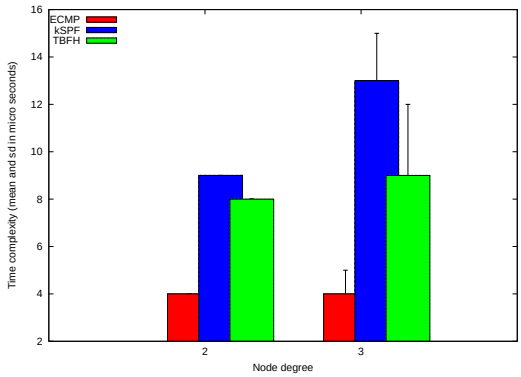
In practice, a kSPF algorithm should be used after an initial SPT computation rooted at the calculating node. It allows to first compute the set of primary next hops to shorten the IGP convergence time, and then compute and validate LFA and/or DC next hops in background. Those two phases should be uncoupled in order to minimize the primary next hop activation time. Thus, a kSPF algorithm requires in practice $(deg(s)+1)$ SPF computations whereas, by design, TBFH uncouples those two phases: TBFH-1 and TBFH-2. Consequently, for kSPF, we have added in our measures the computation time necessary to perform the initial SPT computation.

We observe that the computation time of TBFH does not depend on the node degree whereas the computation time of kSPF increases linearily with $k = deg(s)$. In particular, we observe that the additional computation time necessary for TBFH-2 is relatively small compared to a single SPF computation (ECMP). For example, in ISP6, for nodes having a degree of 10, on average, a single SPF computation requires about 1ms while TBFH adds only 1.5ms. In comparison, kSPF requires almost 10ms. For nodes having the largest degree (28 neighbors), the computation of kSPF time is greater than 30 ms on average.

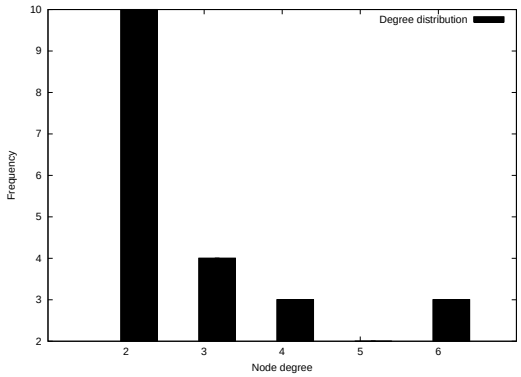
Furthermore, for some routers having a large degree (see for example the router of degree 12 in the Telstra network), the computation time seems to be very low because they are connected to leaf nodes. When this is the case, our improved version of kSPF is really useful in order to scale with the number of neighbors which are connected to the main connected component of the network (even with the removal of the calculating node).



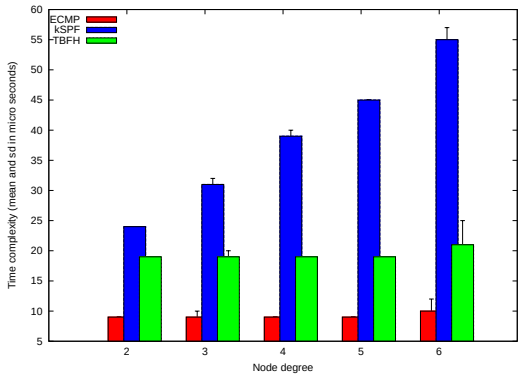
Abilene (11,28) - Degree distribution



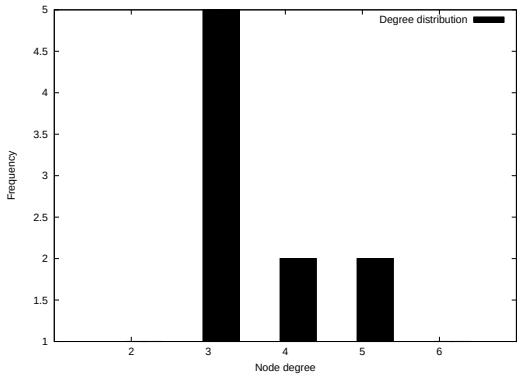
Computational time (μs)



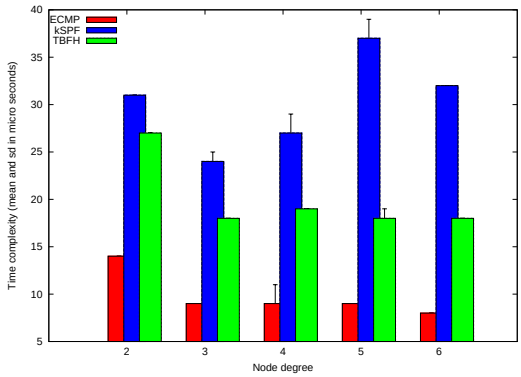
Geant (22,72) - Degree distribution



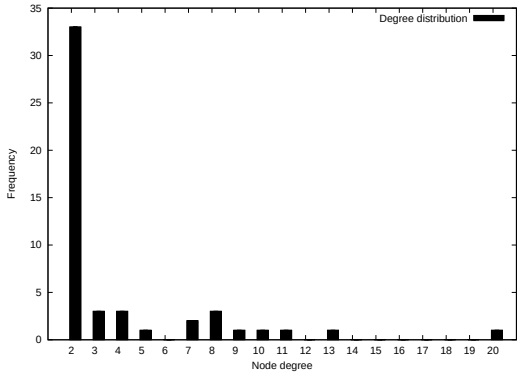
Computational time (μs)



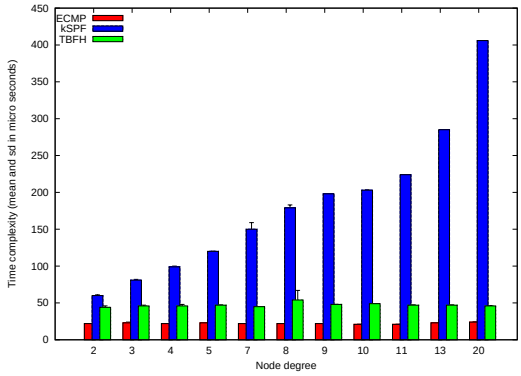
ISP1 (25,50) - Degree distribution



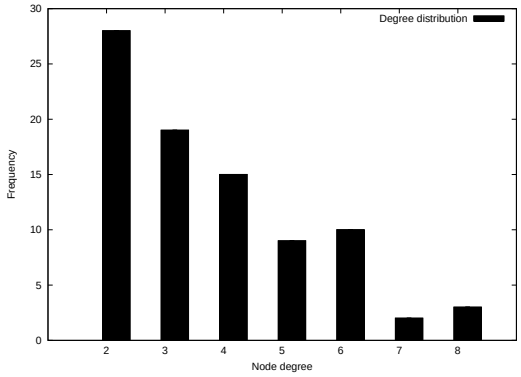
Computational time (μs)



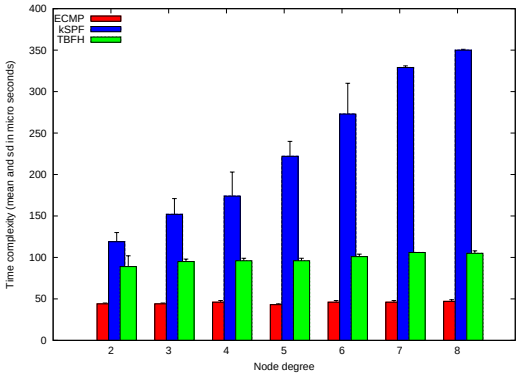
ISP2 (55,200) - Degree distribution



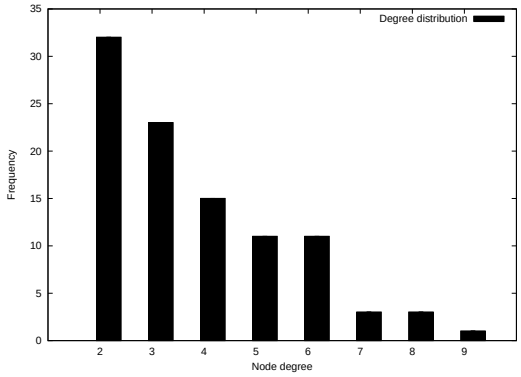
Computational time (μs)



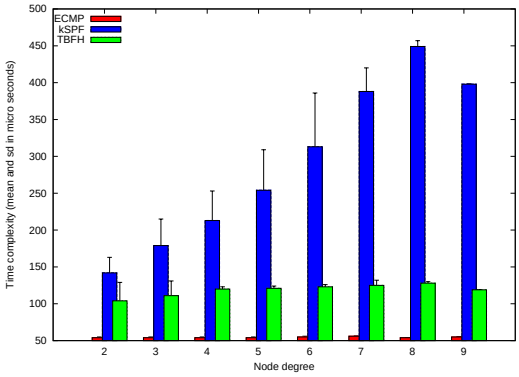
ISP3 (110,350) - Degree distribution



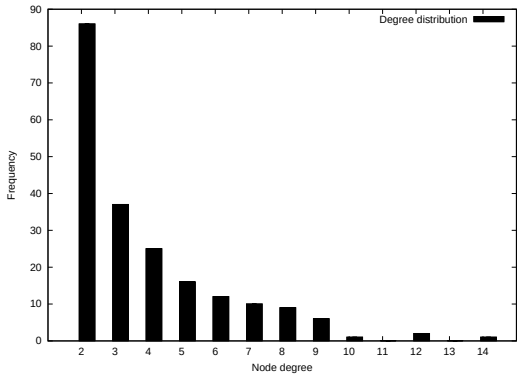
Computational time (μs)



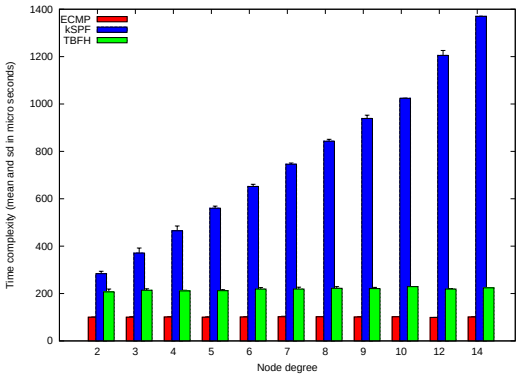
ISP4 (140,410) - Degree distribution



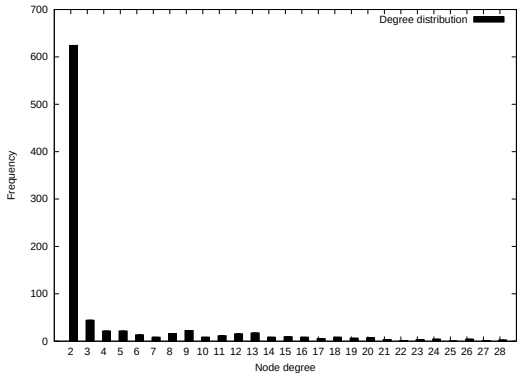
Computational time (μs)



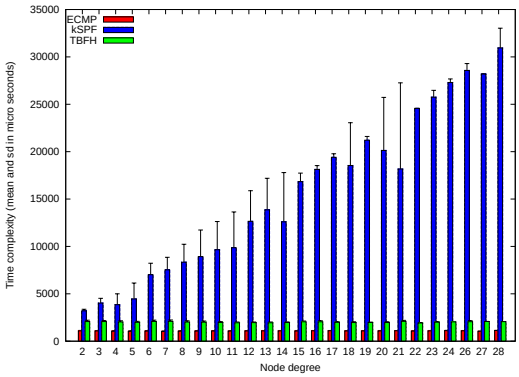
ISP5 (210,800) - Degree distribution



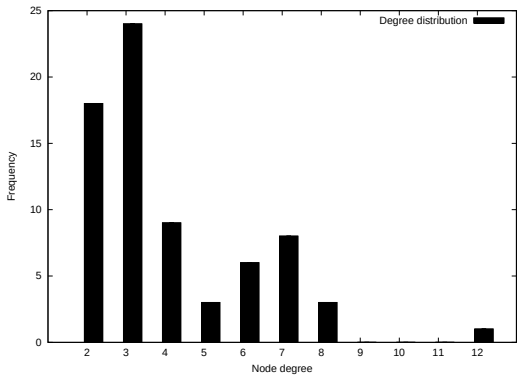
Computational time (μs)



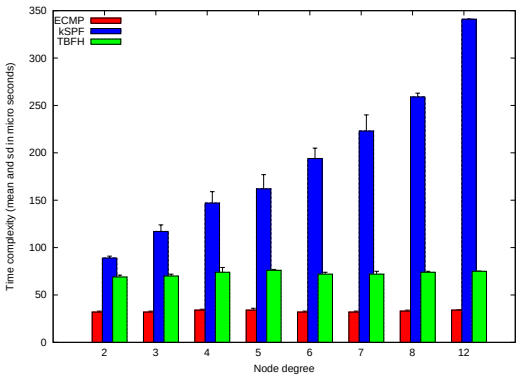
ISP6 (1170,4200) - Degree distribution



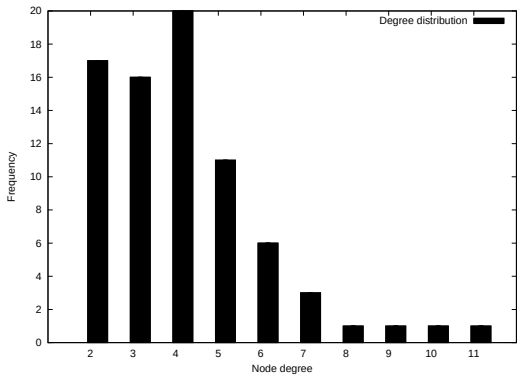
Computational time (μs)



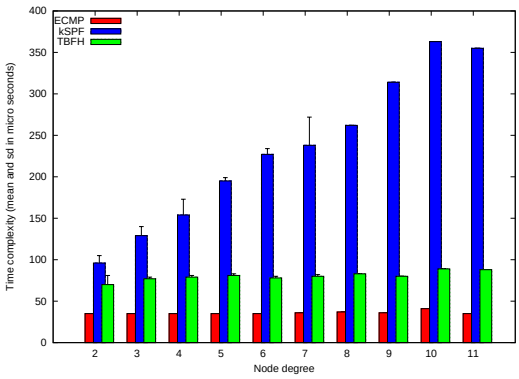
Exodus (79,294) - Degree distribution



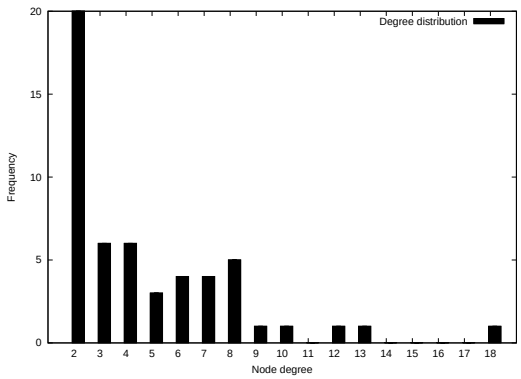
Computational time (μs)



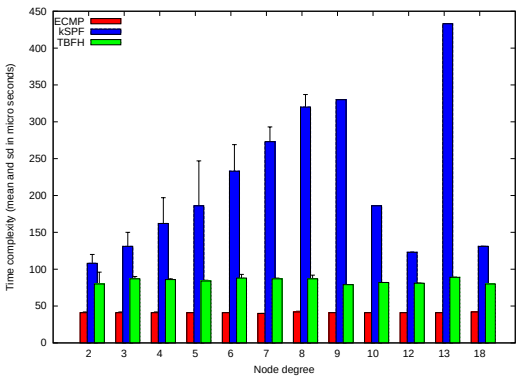
Ebone (87,322) - Degree distribution



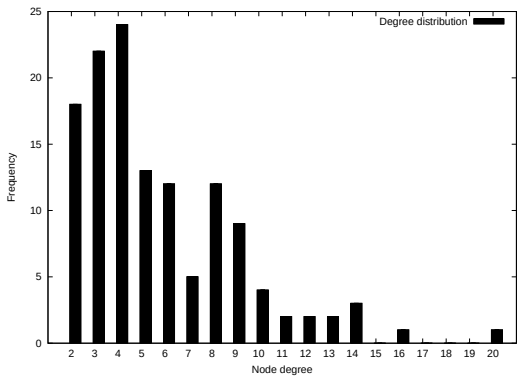
Computational time (μs)



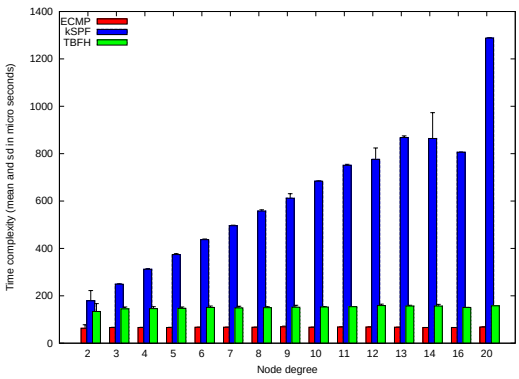
Telstra (108,306) - Degree distribution



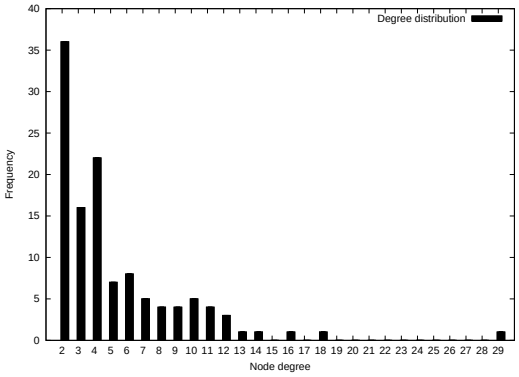
Computational time (μs)



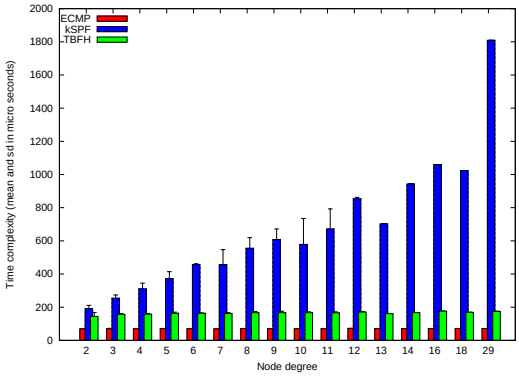
AboveNet (141,748) - Degree distribution



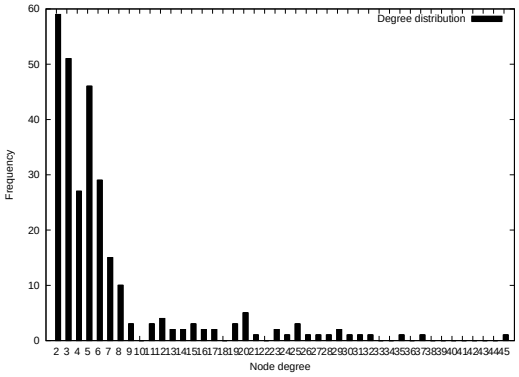
Computational time (μs)



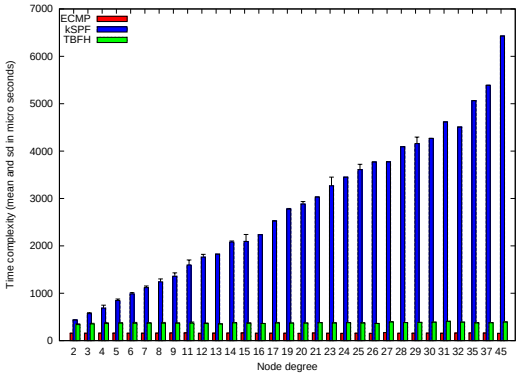
Tiscali (161,656) - Degree distribution



Computational time (μs)



Sprint (315,1944) - Degree distribution



Computational time (μs)

TABLE III: Computional times per node degree (and respective node degree distributions)

The time complexity gain provided by TBFH increases according to the degree of the calculating node and the graph size. Indeed, the gain for smaller topologies such as ISP3 is comparatively lower because it is simply proportional. Our algorithm is designed for large topologies and bounds the computational overhead of each router to less than one additional SPF. TBFH is an efficient algorithm to limit the time complexity overhead for nodes having large degrees.Table IV gives the average computation time of kSPF and TBFH for all actual and inferred topologies of our set of evaluation networks.

C. Coverage and Path Diversity Results

In this section, we focus on the *coverage*, e.g., the ability to provide at least two forwarding choices, of several routing and algorithmic approaches. In practice, we evaluate the coverage of TBFH compared to the optimal one provided by a kSPF algorithm. Generally, the notion of coverage is used to evaluate the capacity of protection provided by fast reroute techniques. However, we also use it to evaluate the forwarding diversity in a load balancing context. Indeed, for load balancing, the coverage allows us to measure the forwarding diversity using a *power of two* perspective [31]. We argue that this measure is more meaningful than computing an average number of loop-free next hops because it allows to know the fraction of pairs (router,destination) which can (or not) take advantage of at least two next hops.

For a graph G and according to a path computation algorithm alg and a loop-free rule r, the coverage $cov(G, alg, r)$ is computed as the following ratio:

$$cov(G, alg, r) = \frac{\sum_{\forall (n,\ d) \in N} vNH(n, d)}{|N| \times (|N| - 1)} \quad (7)$$

where $vNH(n, d)$ is equal to 1 if the algorithm alg computes at least one valid loop-free next hop using rule r on the node n towards the destination d, otherwise $vNH(n, d) = 0$.

Table IV summarizes the characteristics of our set of actual and inferred networks and highlights the tradeoff between coverage and computation time.
We have analysed the number of pairs of routers which can benefit from ECMP, DC and LFA. We organize Table IV according to the multipath objective: with ECMP and DC, alternate next hops can be used for both fast re-routing and load balancing purposes (the MP set) whereas LFA next hops are used only in case of failure (the FRR set).

In this first study, the term coverage is local and given with formula (7). We measure the percentage of pairs of routers taking advantage of at least one valid alternate path, a kSPF algorithm providing the upper bound: the optimum value (denoted *optimal* in Table IV). Our main concern is to analyze the ability of TBFH and TBFH" to achieve a coverage close to the optimum. Indeed, TBFH or TBFH" might miss some valid next hops because it focuses on optimal 1-alternate paths and their post convergence resulting next hops.

Table IV provides a global overview of DC and LFA coverages for each evaluation topology. We observe that, in most cases, DC or LFA next hops are also local post convergence next hops: the coverage capacity is only slightly impacted by focusing on post convergence next hops. Indeed, coverage results of TBFH are very close to the upper bound (using kSPF). The greatest difference is about 7% for Ebone and ISP5 networks using the LFA rule, and lower than 4% on average for the DC rule (the worst case is for ISP5 with 9%).

This low difference is particularly interesting considering the LFA rule because next hops guaranteeing the local post convergence property ensure a stable re-routing scheme (see Sec. IV-B). We can also notice that the standard deviation around the mean of the computation time required by TBFH is very low. Indeed, TBFH does not depend on the degree of the calculating node whereas kSPF depends on it by definition. On the contrary, using kSPF for large IP networks such as ISP6, the standard deviation is very high : $\sim 5000\mu$s for kSPF (instead of $\sim 150\mu$s for TBFH).

For TBFH", note that we only select LFA next hops which can be used after the IGP convergence. Either they are local post convergence LFA according to Definition 1 or they verify the DC rule for the post convergence best cost $C_2(s, d)$. Indeed, it means that they can be used in the MP set after the IGP convergence without generating path flapping. In practice, TBFH" can validate more LFA next hops if we do consider this property. However, we can observe that TBFH" provides an excellent tradeoff between coverage and computation time as highlighted in Table IV. For the DC rule, the worst case is ISP3 with a difference of 3%. For the LFA rule, the biggest difference is 6% for ISP6: it means that the remaining 6% of LFA next hops do not ensure the post convergence property.

In order to quantify the coverage as an end to end measure, we have also computed an end to end protection ratio. Those results indicate the probability of coverage for each primary path, i.e., the probability of having an alternate forwarding path not using a given link belonging to the primary route. For the DC rule and for each pair of nodes, the set of alternate routes forms a directed acyclic graph (DAG): using the DC rule, load balancing next hops are distributively composed to form a DAG per pair (src, dst). For all primary paths having the same number of hops[9], we compute an end to end coverage ratio. This ratio quantifies the average number of links belonging to the primary path which can be circumvented thanks to one or several alternate forwarding paths in the DAG. To achieve this ability, either we must consider a back-pressure mechanism such as the one described in [20], or use a path selector field to allow sources to explore all the DAG.

For the LFA rule, the measure has the same sense but the computation method is different. Indeed, for a given pair (src, dst), several LFA next hops cannot be simultaneously used otherwise their composition may form cycles (the LFA rule only supports single link failure, otherwise forwarding loops may appear). At each hop of the primary path, we check

[9]Note that primary paths have been computed according to links weight.

TABLE IV
LOCAL COVERAGE AND COMPUTATION TIME ON REAL AND INFERRED TOPOLOGIES

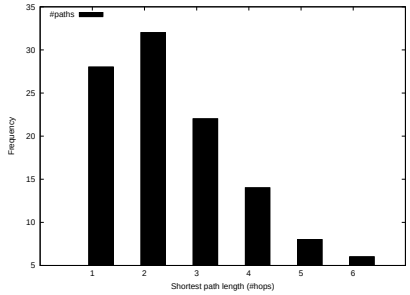
Network name	Size		**Load Balancing** Coverage (%)				**Fast Re-Routing** Coverage(%)			**Computation Time (μs)** mean & standard deviation			
			ECMP	DC			LFA						
	N	E		**TBFH**	**TBFH"**	optimal	**TBFH**	**TBFH"**	optimal	ECMP	**TBFH**	**TBFH"**	kSPF
Abilene	11	28	0	39	39	39	61	65	65	4 ± 1	8 ± 1	9 ± 3	11 ± 2
Geant	22	72	0	69	69	69	86	86	89	9 ± 1	18 ± 1	22 ± 3	34 ± 11
ISP1	25	50	0	5	5	5	19	19	19	9 ± 1	17 ± 1	22 ± 2	30 ± 5
ISP2	55	200	3	49	49	49	83	83	84	22 ± 1	43 ± 1	53 ± 11	99 ± 72
ISP3	110	350	10	34	36	39	57	58	61	46 ± 1	90 ± 8	113 ± 19	188 ± 71
ISP4	140	410	8	28	30	32	48	49	51	56 ± 1	104 ± 15	136 ± 30	219 ± 90
ISP5	210	800	20	56	64	65	74	77	81	105 ± 2	199 ± 9	252 ± 43	473 ± 225
ISP6	1170	4200	22	54	56	57	64	64	70	1105 ± 49	1987 ± 160	2292 ± 503	5555 ± 5288
Exodus	79	294	16	47	49	50	70	72	75	34 ± 1	69 ± 3	90 ± 14	153 ± 60
Ebone	87	322	17	42	44	45	64	66	71	37 ± 1	75 ± 6	97 ± 17	174 ± 65
Telstra	108	306	7	21	21	21	37	38	39	42 ± 1	81 ± 9	104 ± 22	185 ± 89
AboveNet	141	748	17	62	65	66	82	85	87	66 ± 5	137 ± 13	212 ± 51	410 ± 201
Tiscali	161	656	13	42	44	44	58	59	62	71 ± 2	143 ± 14	207 ± 59	384 ± 246
Sprint	315	1944	27	61	65	65	85	86	86	159 ± 3	335 ± 15	554 ± 249	1126 ± 1010

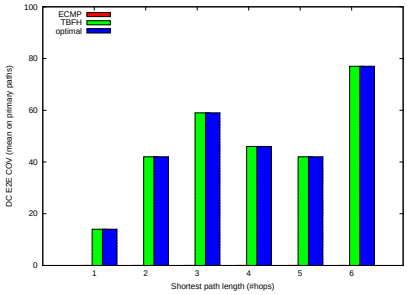
if a LFA next hop exists or not, and then we measure the average number of links which can be covered with a local LFA along the primary path. Plots given in Table V use the same x-axis: the number of links on the primary paths. Table V(a) provides the primary paths distribution in terms of hops number.

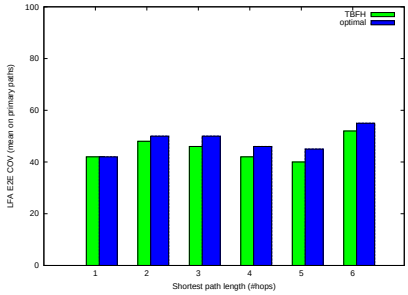
Table V(b) provides the ECMP and TBFH end to end coverage considering the DC rule compared to the upper bound provided by a kSPF algorithm (optimal). Table V(c) plots the TBFH end to end coverage considering the LFA rule. These plots allow to analyze coverage results given in Table IV more deeply.

For the LFA rule applied to ISP5 (the worst case), we notice that links which are LFA covered seem to be less used by primary routes than links which are not. Indeed, the average coverage of almost 76% given in Table VI-C is lower than the 81% provided in Table IV: this means that less primary routes goes through LFA covered links than through non covered links. The LFA and DC coverage distribution obviously depends on the topology: we do not observe this loss of LFA coverage in all topologies as shown in Tables V and VI-C.

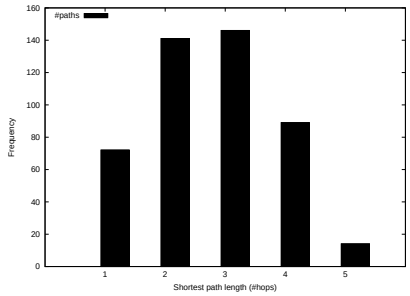
For the DC rule, end to end coverage optimal results are better than the local ones given in Table IV because we consider the possibility to balance the load anywhere along the DAG (not only at the router with the failed/congested link). In practice, for ISP5, the increase of coverage is about 15% on average (DC achieves an end to end coverage of 79%) meaning that DC is able to significantly cover these networks if we consider the possibility to notify a failure or a congestion upstream to the detected outage. Indeed, if packets can be deflected upstream to the congested (or failing) link, the DAG is able to cover almost all the links used in the path (in particular, for long paths). We can also remark that short paths are not well protected using the DC rule because destination nodes are too close to find valid alternate paths. Note that if there exists a DC or LFA next hop covering the destination node corresponding to the head of a given link, then all destinations using this link through their primary path are also covered. This property is respectively called per-link DC and per-link LFA: it denotes the case where all destinations using a given primary next hop are protected.

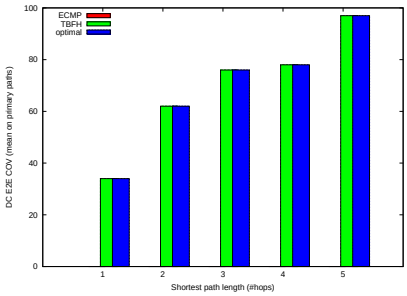


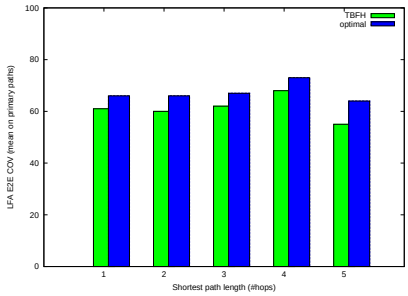




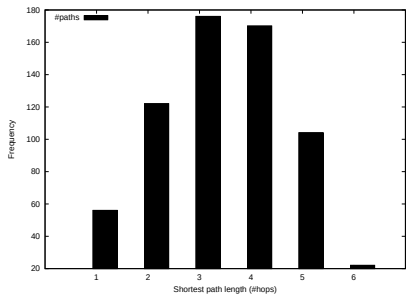
Abilene - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

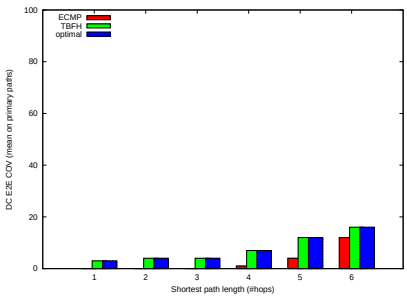


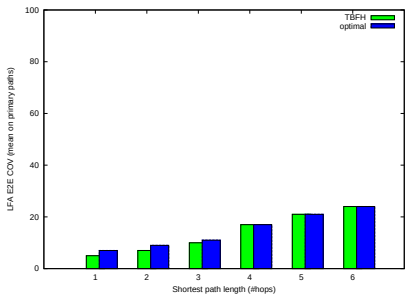




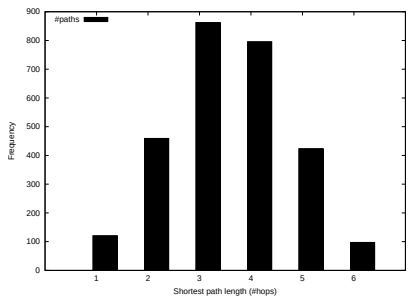
Geant - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

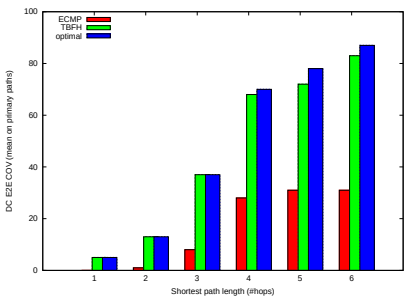


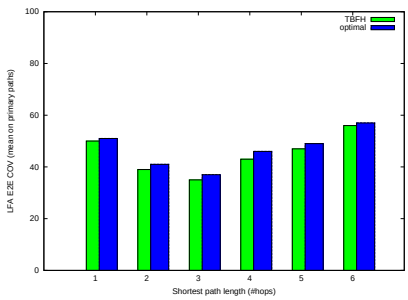




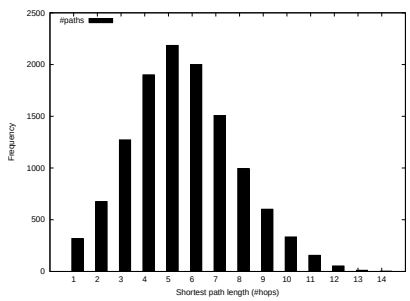
ISP1 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

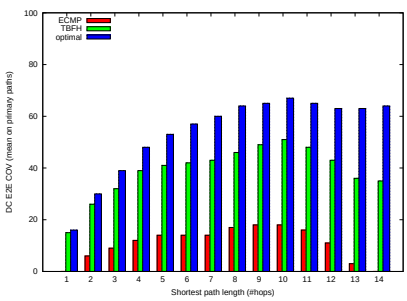


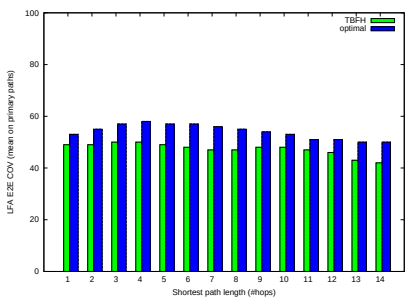




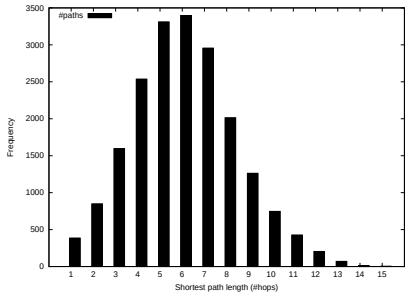
ISP2 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

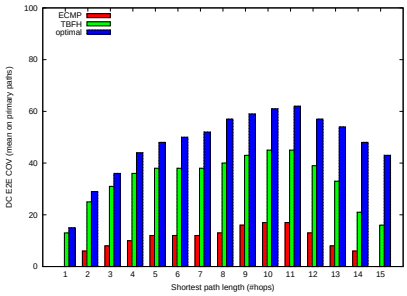


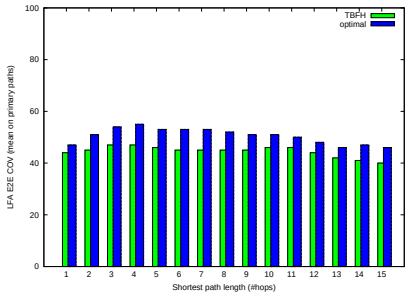




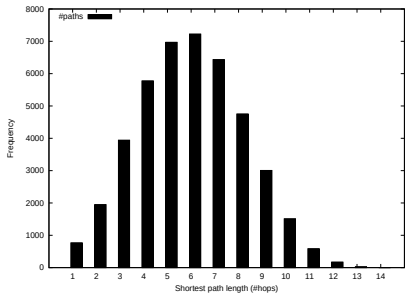
ISP3 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

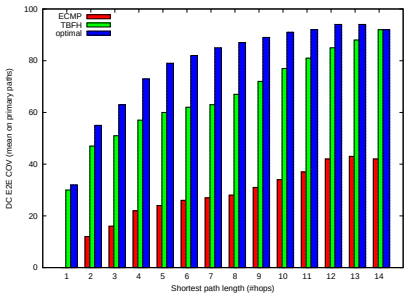


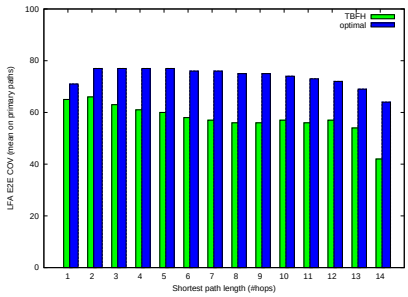




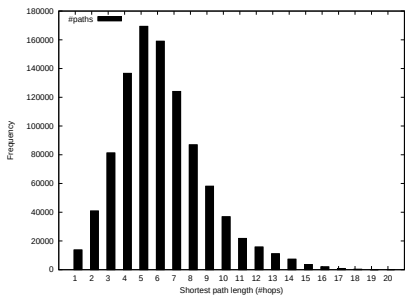
ISP4 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

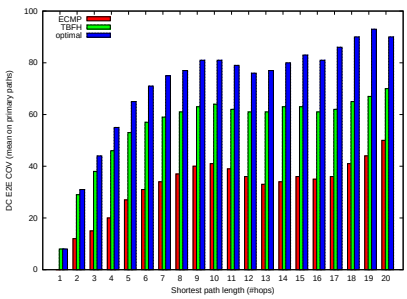


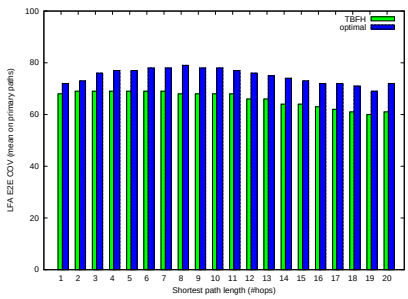




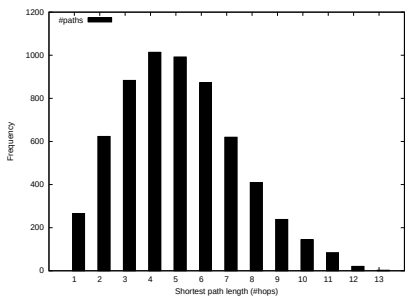
ISP5 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

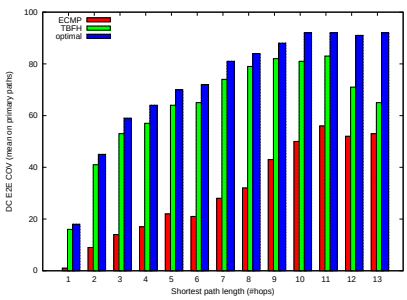


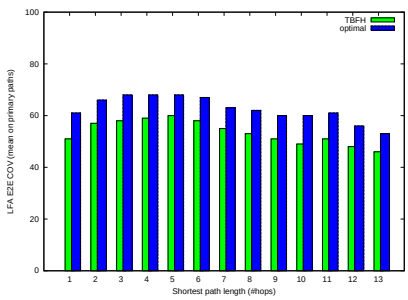




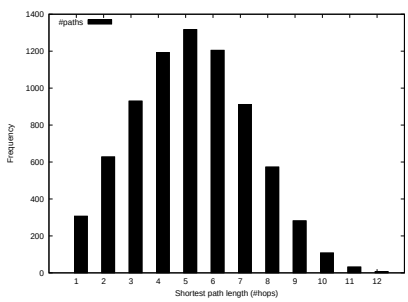
ISP6 - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

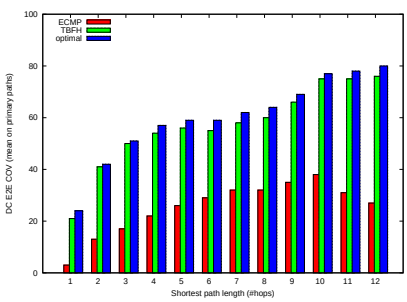


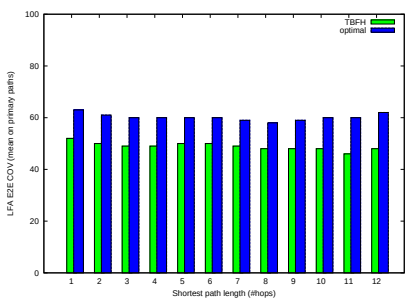




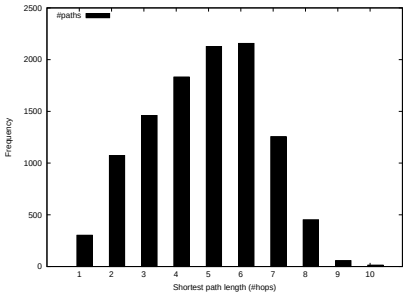
Exodus - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

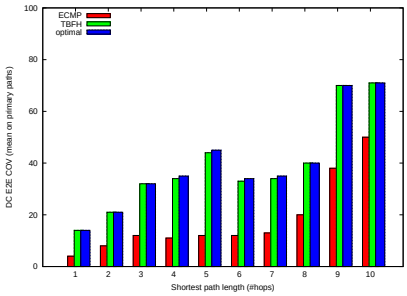


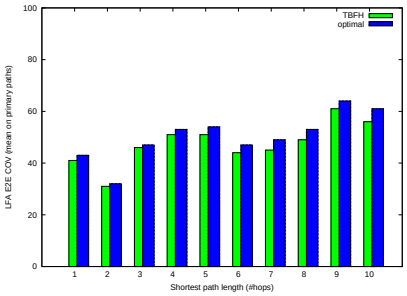




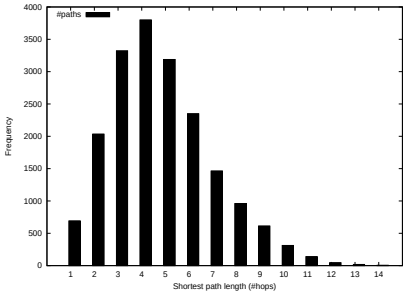
Ebone - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

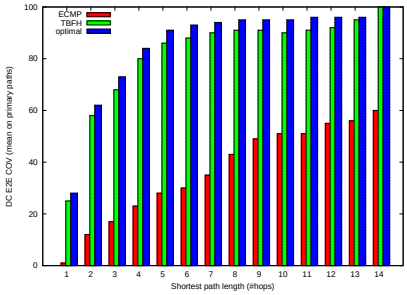


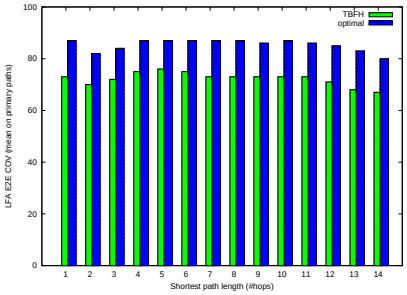




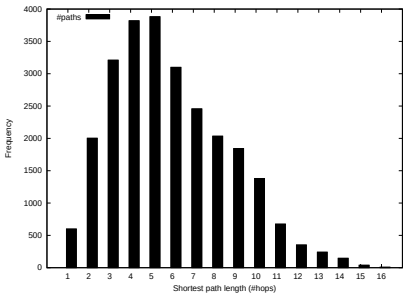
Telstra - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

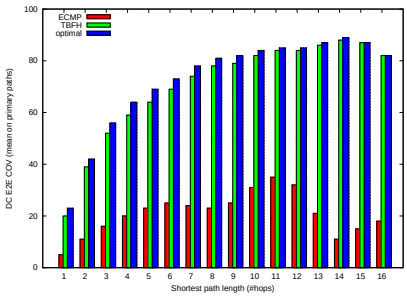


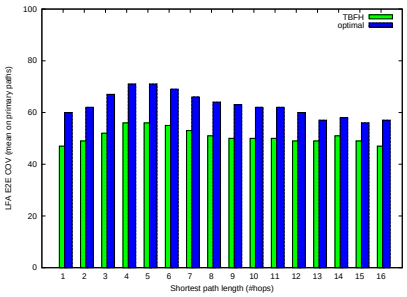




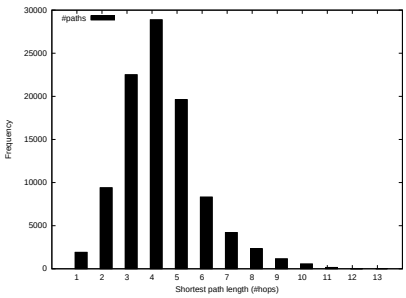
AboveNet - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

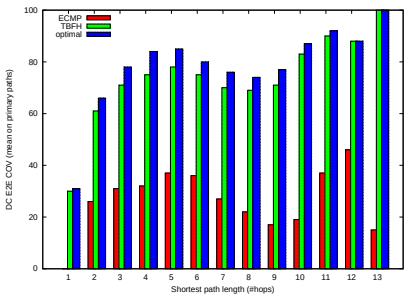


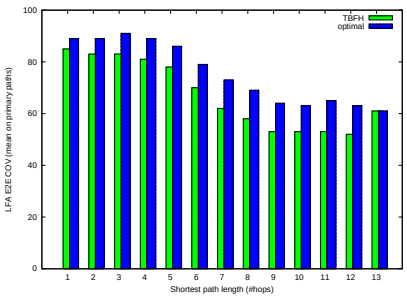




Tiscali - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage







Sprint - (a) Primary paths distribution (b) DC E2E Coverage (c) LFA E2E Coverage

TABLE V: End to end coverage results of TBFH (per primary path length)

For the DC rule, the difference seems even more significant than with LFA because alternate next hops can be composed all along the DAG. However, the difference with the optimum remains low compared to ECMP: in the worst case for the DC rule (ISP5), the loss is lower than 16% on average (instead of more than 50% with ECMP). For the LFA rule and considering the worst case, the difference is always lower than 20% whatever the length of the primary path.

Table V also analyzes whether the loss of coverage of TBFH seems more significant using an end to end perspective. This series of plots intends to show cases where TBFH (and ECMP) miss valid alternate next hops. The difference of coverage is greater than in Table IV meaning that the loss is not uniformly distributed. Although the difference of coverage appears greater than in Table IV, these results allow to emphasize limitations of the basic version of TBFH: (i) it only computes two next hops per destination (and that can lead to a loss of path diversity when used with the DC rule[10]), (ii) it may miss valid next hops located at key positions in the network. More precisely, candidate next hops which are not computed with TBFH do not seem uniformly distributed across the network: links which are not covered by TBFH seem to be a little bit more useful than covered ones (in terms of number of primary routes using them). In practice, we observe that the loss of coverage caused by TBFH seems to be due to links located in the core of the network.

For example, compared to kSPF, the average loss of coverage due to TBFH is about 15% for 6 hops paths whereas the local loss coverage is lower. With TBFH", this average loss is reduced to less than 10%. LFA next hops which are not computed by TBFH are not uniformly distributed across the network: links which are not covered with a post convergence next hop are more used than covered ones (in terms of number of primary routes using them). Indeed, such a loss of coverage compared to results given in Table IV implies that the TBFH ability to compute alternate next hops is not uniformly distributed accross the network. However, this difference remains relatively low. In practice, we observe that TBFH" is able to provide a very good DC coverage whereas the loss of coverage for LFA next hops is mainly due to the post convergence property.

Compared to ECMP, TBFH and TBFH" provide excellent coverage results. TBFH only requires a complexity equivalent to ECMP in terms of computation and deployment while the time complexity overhead of TBFH" is very limited compared to kSPF. Although the computation time remains low as shown in Table VII(a), the DC and LFA coverage provided by TBFH" is closer to the optimal. As highlighted in Table VII(b) and (c), TBFH" provides a good tradeoff between coverage and time complexity: there is only a slight dependence on the degree of the calculating node.

Table VI-C also emphasizes the difference between the end-to-end coverage and the local one (given in Table IV). Using this perspective, the gain of coverage provided by TFBH" appears more significant than using the local coverage perspective. However, we can notice that the second best next hop (provided by TBFH) is generally sufficient to provide the best DC or LFA loop-free next hop. Indeed, in the majority of the cases, the coverage is ensured by post convergence next hops.

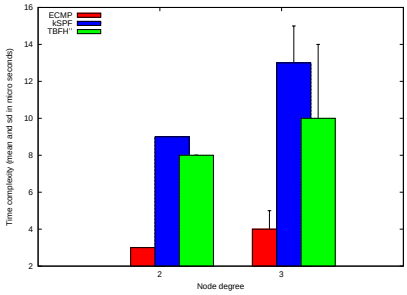
Using TBFH", the extension of TBFH requiring an additional time complexity of $deg(s)\ |E|$, the coverage can be strongly improved. Indeed, although the computation time remains low, the DC and LFA coverage are closer to the optimal. TBFH" is able to compute more next hops than TBFH and this difference is more significant using and end to end perspective: additional next hops are composed between neighbor routers to form more routes. This improvement explains why TBFH" gives better results than TBFH using a metric taking into account all the path diversity. Moreover, it allows routers to compute valid alternate next hops located at the core of the network (the ones which are the more used in terms of the number of routes going through them). It is also worth to notice that the TVFH algorithm given in Sec. **??** is able to compute at least one valid alternate next hop if necessary. The variants of TBFH allows routers to perform a good tradeoff between alternate path optimality and end to end coverage.

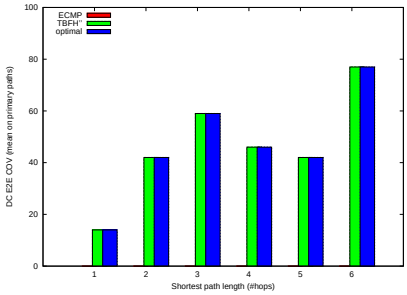
To summarize, we have seen that TBFH and even more TBFH" provide excellent coverage results, close to optimal ones (using a kSPF algorithm). TBFH and TBFH" also ensure the post convergence property for LFA next hops while their time complexity remains very low as shown in Sec. VI-B and Sec. VI-C. Compared to ECMP, our algorithms strongly improve the diversity of forwarding paths without leading to a significant computation and deployment overhead. Finally, its set of algorithmic variants allow one to provide a modular and extensible framework for fast re-routing and load balancing purpose.

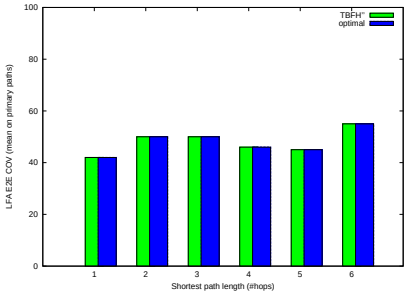
[10]For the DC rule, the difference may seem even more significant because alternate next hops can be composed all along the DAG.

Network Name		DC rule			LFA rule		
	ECMP	**TBFH**	**TBFH”**	optimal	**TBFH**	**TBFH”**	optimal
Abilene	0.00	40.94	40.94	40.94	45.55	47.57	47.57
Geant	0.00	66.71	66.71	66.71	62.52	63.53	67.99
ISP1	1.45	6.81	6.81	6.81	13.54	13.54	14.51
ISP2	17.11	48.01	49.89	49.89	41.26	41.47	43.97
ISP3	13.13	40.27	47.51	52.36	48.85	50.69	56.70
ISP4	12.14	37.33	44.37	48.75	45.82	47.48	52.82
ISP5	24.81	61.12	77.32	78.81	59.29	68.18	76.18
ISP6	28.69	53.07	62.61	65.69	68.46	73.01	77.38
Exodus	21.45	60.59	65.55	67.06	57.41	60.42	66.23
Ebone	24.97	53.67	55.38	56.66	49.65	53.02	60.27
Telstra	12.21	34.43	34.97	35.04	46.15	47.05	48.57
AboveNet	25.48	77.74	81.49	82.33	74.24	80.51	86.32
Tiscali	21.81	65.02	67.59	68.90	53.10	56.81	66.81
Sprint	31.66	72.41	78.63	79.54	78.87	83.46	86.58

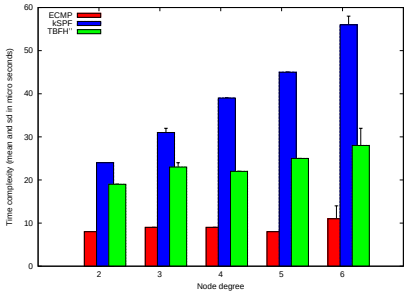
TABLE VI
END TO END COVERAGE AVERAGE RESULTS FOR TBFH AND TBFH”

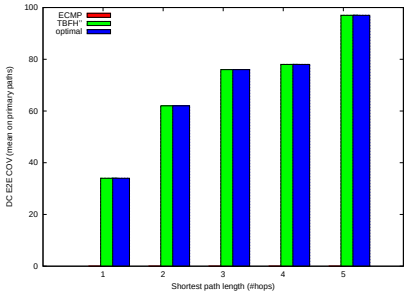


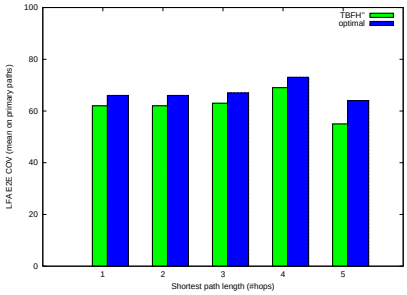




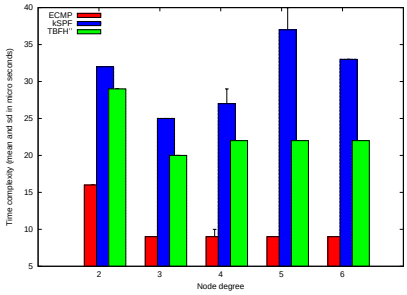
Abilene - (a) TBFH” Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

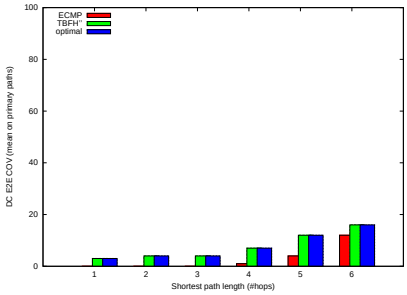


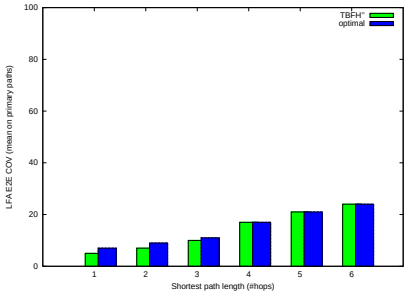




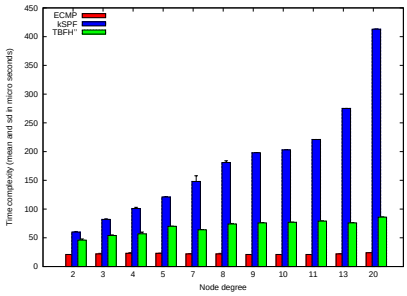
Geant - (a) TBFH” Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

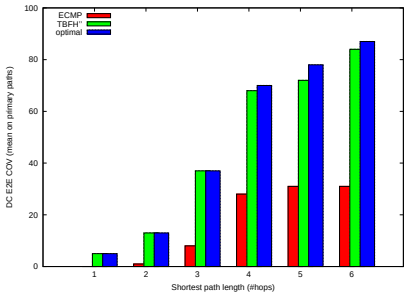


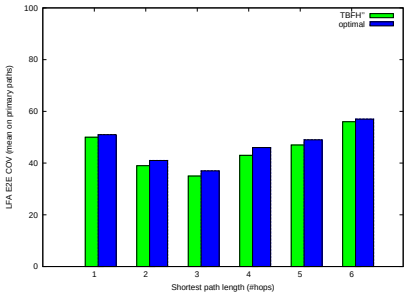




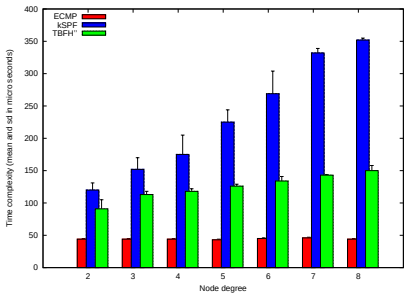
ISP1 - (a) TBFH” Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

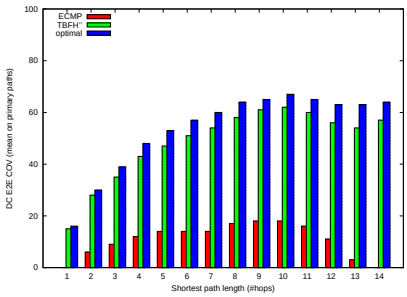


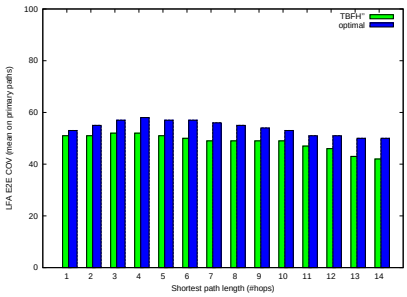




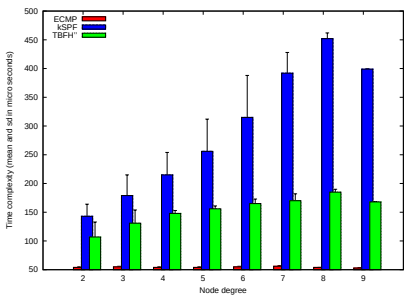
ISP2 - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

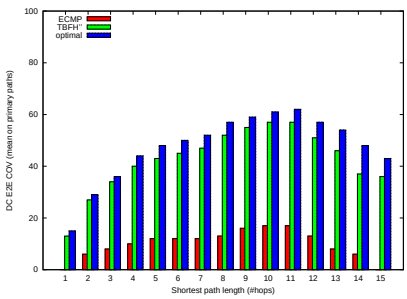


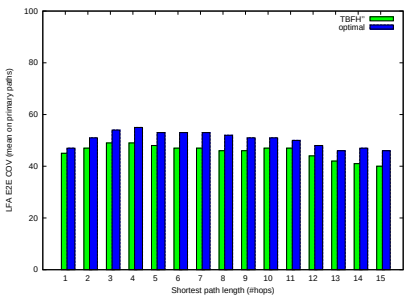




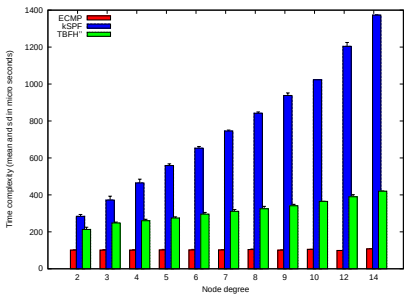
ISP3 - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

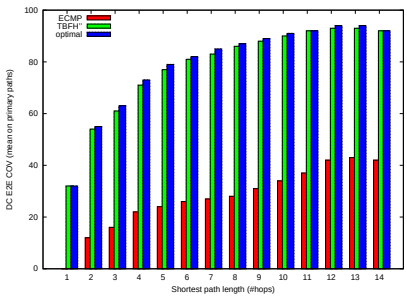


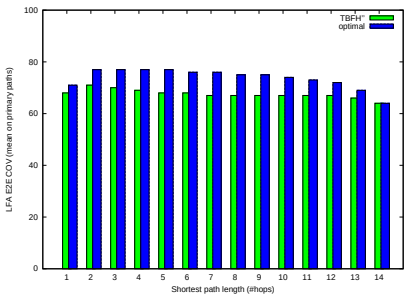




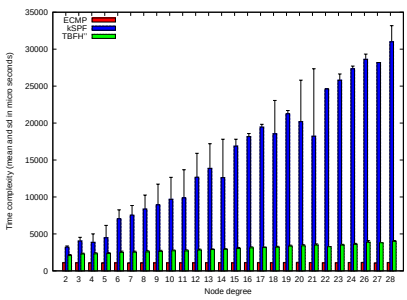
ISP4 - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

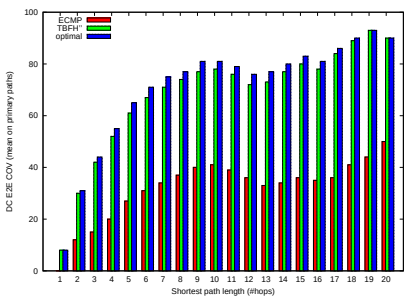


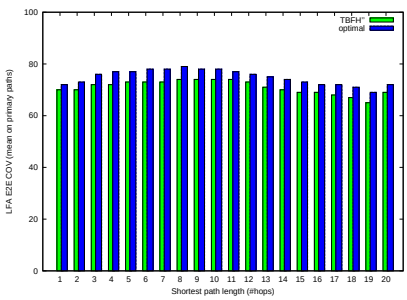




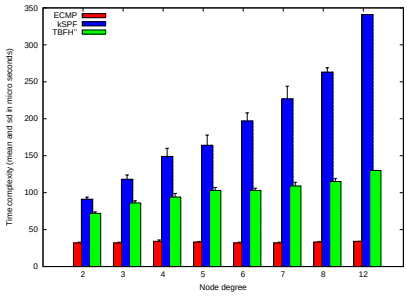
ISP5 - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

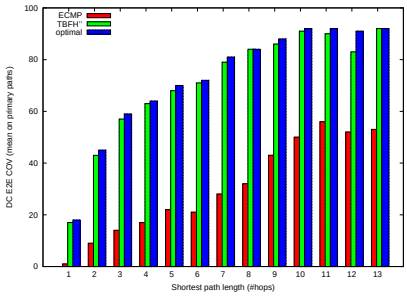


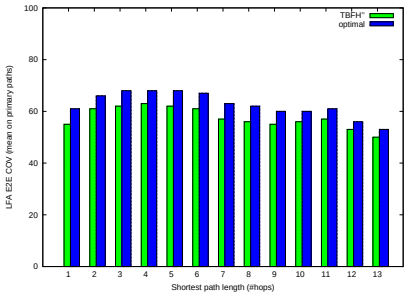




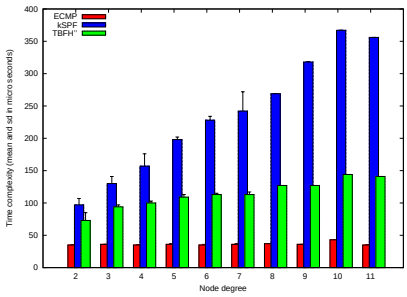
ISP6 - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

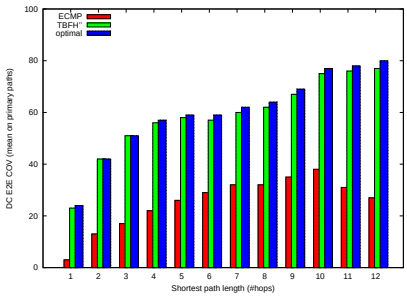


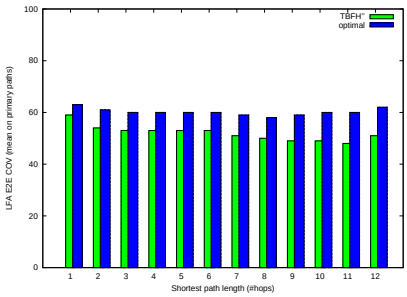




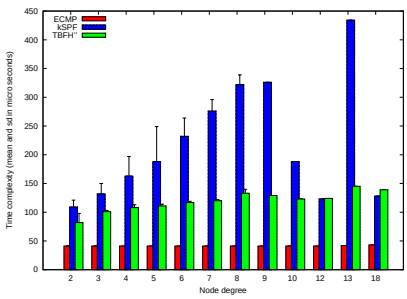
Exodus - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

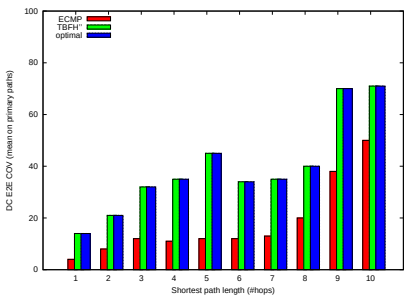


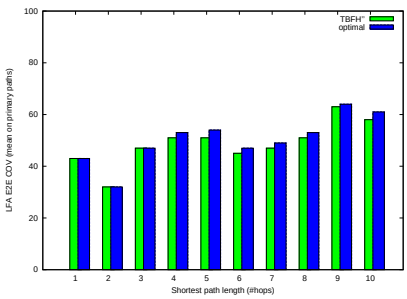




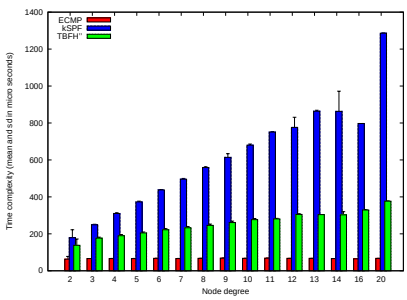
Ebone - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

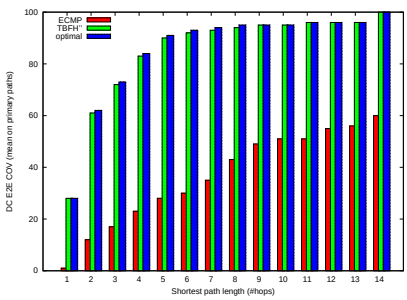


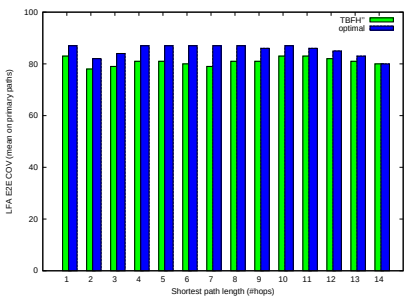




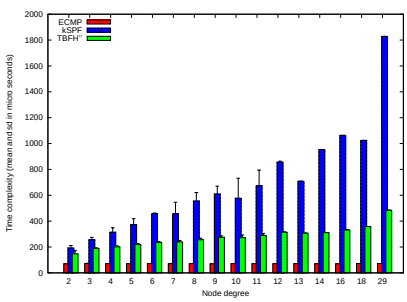
Telstra - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

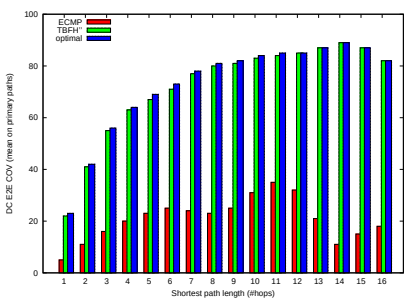


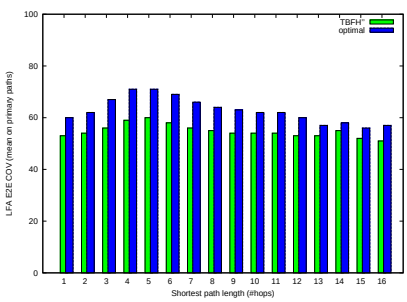




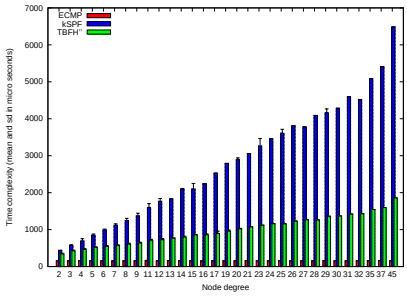
AboveNet - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

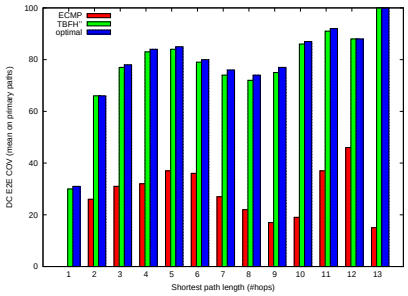


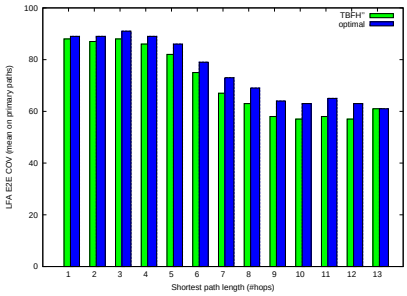




Tiscali - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage







Sprint - (a) TBFH" Computation time (b) DC E2E Coverage (c) LFA E2E Coverage

TABLE VII: End to end coverage result of TBFH"

VII. Conclusion

Multipath routing enhances the network reliability: it allows for load balancing and fast re-routing to circumvent congestions and failures. However, the overhead imposed by signaling messages, the time and space complexity of multiple routes computation can hamper its deployment.

In this paper, we propose an efficient algorithm, TBFH, which provides a greater path diversity than ECMP with a very low overhead. In particular, TBFH efficiently computes the two best first hop disjoint paths. The time complexity of TBFH does not depend on the degree of the calculating router.

Furthermore, we propose a general multipath forwarding scheme that provides load balancing and fast re-routing next hops. One possible application of this scheme is to expose the forwarding diversity to the end hosts and allow them to control load shifting decisions thanks to a tagging mechanism. We have considered two validation rules ensuring loop-free forwarding, the downstream criterion for load balancing, and the loop-free alternate rule for local fast recovery. In a hop by hop forwarding context, the alternate next hops computed with TBFH are called local post convergence next hops. This set of next hops minimizes the number of flow deflections in case of failure.

Using a large set of topologies, we have shown that local post convergence next hops are in more than 90% of the cases the best loop-free alternate next hops. Our proposition can be incrementally integrated in OSPF or IS-IS by replacing the path computation algorithm without any additional messages in the control plane. Our solution is scalable for large IP networks with high degree routers and can operate in conjunction with routers using ECMP as well with non multipath-capable ones.

Acknowledgement

This work is partially funded by Trilogy, a research project (ICT-216372) supported by the European Community under its Seventh Framework Programme. The views expressed here are those of the authors only.
P. Francois is supported by the FNRS (Fonds National de la Recherche Scientifique, Belgium).

References

[1] "ISO/IEC 10589 Information technology Telecommunications and information exchange between systems Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service," ISO 8473.

[2] "OSPF Incremental SPF," http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/ospfispf.html.

[3] "A tool for path computation algorithms," http://inl.info.ucl.ac.be/content/tool-path-computation-algorithms.

[4] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A Measurement-based Analysis of Multihoming," in *SIGCOMM*, 2003.

[5] D. Applegate and E. Cohen, "Making Intra-domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs," in *SIGCOMM*, 2003.

[6] A. Atlas and A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates," IETF, RFC 5286, 2008.

[7] R. Banner and A. Orda, "Multipath Routing Algorithms for Congestion Minimization," *IEEE/ACM Trans. Netw.*, 2007.

[8] S. Bryant, C. Filsfils, S. Previdi, and M. Shand, "IP Fast Reroute using Tunnels, draft-bryant-ipfrr-tunnels-03," IETF, Draft, November 2007.

[9] S. Bryant, M. Shand, and S. Previdi, "IP Fast Reroute Using Not-via Addresses draft-bryant-shand-ipfrr-notvia-addresses-03.txt," IETF, Draft, Oct. 2006.

[10] J. Chen, P. Druschel, and D. Subramanian, "An Efficient Multipath Forwarding Method," *IEEE INFOCOM*, 1998.

[11] I. Cidon, R. Rom, and Y. Shavitt, "Analysis of Multi-path Routing," *IEEE/ACM Trans. Netw.*, 1999.

[12] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[13] E. Dijkstra, "A Note on Two Problems in Connection with Graphs," *In Numerische Mathematik*, 1959.

[14] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *INFOCOM*, 2001.

[15] D. Eppstein, "Bibliography on k Shortest Paths and other k Best Solutions Problems," http://www.ics.uci.edu/~eppstein/bibs/kpath.bib.

[16] D. Eppstein, "Finding the k Shortest Paths," in *IEEE Symposium on Foundations of Computer Science*, 1994.

[17] C. Filsfils, "BGP Convergence in Much Less than a Second," http://www.nanog.org/meetings/nanog40/presentations/ClarenceFilsfils-BGP.pdf.

[18] P. Francois and O. Bonaventure, "An Evaluation of IP-based Fast Reroute Techniques," in *CoNEXT*, 2005.

[19] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP Convergence in Large IP Networks," *SIGCOMM*, vol. 35, no. 3, pp. 35–44, 2005.

[20] I. Gojmerac, P. Reichl, and L. Jansen, "Towards Low-complexity Internet Traffic Engineering: The Adaptive Multi-Path Algorithm," *Comput. Netw.*, 2008.

[21] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," IETF, RFC 2992, Nov. 2000.

[22] H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," *Wirel. Netw.*, vol. 11, no. 1-2, pp. 99–114, 2005.

[23] G. Iannaccone and al., "Feasibility of IP Restoration in a Tier-1 Backbone," *IEEE Networks Magazine*, March 2004.

[24] J. R. Iyengar, P. D. Amer, and R. R. Stewart, "Performance implications of a bounded receive buffer in concurrent multipath transfer." *Computer Communications*, vol. 30, no. 4, pp. 818–829, 2007.

[25] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *SIGCOMM*, 2005.

[26] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi, "BANANAS: an Evolutionary Framework for Explicit and Multipath Routing in the Internet," *SIGCOMM*, 2003.

[27] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring Link Weights using End-to-End Measurements," in *ACM SIGCOMM Internet Measurement Workshop*, 2002.

[28] J. M. McQuillan, I. Richer, , and E. C. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, 1980.

[29] P. Mérindol, J.-J. Pansiot, and S. Cateloin, "Improving Load Balancing with Multipath Routing," in *ICCCN*, 2008.

[30] ——, "Low Complexity Link State Multipath Routing," in *Global Internet Symposium (Infocom workshop)*, 2009.

[31] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *IEEE Trans. Parallel Distrib. Syst.*, 2001.

[32] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path Splicing," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 27–38, 2008.

[33] P. Narvaez and K. Y. Siu, "Efficient Algorithms for Multi-Path Link State Routing," in *ISCOM'99*, Kaohsiung, Taiwan, 1999.

[34] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New Dynamic Algorithms for Shortest Path Tree Computation," *IEEE/ACM Trans. Netw.*, 2000.

[35] Y. Ohara, S. Imahori, and R. V. Meter, "MARA: Maximum Alternative Routing Algorithm," *Infocom*, 2009.

[36] M. Pascoal, "Implementations and Empirical Comparison for K Shortest Loopless Path Algorithms," in *The Ninth DIMACS Implementation Challenge: The Shortest Path Problem*, 2006.

[37] B. Quoitin, "Topology Generation through Network Design Heuristics," http://www.info.ucl.ac.be/~bqu/igen/.

[38] B. Quoitin, V. V. den Schrieck, P. Francois, and O. Bonaventure, "IGEN: Generation of Router-level Internet Topologies through Network Design Heuristics," *In Proceedings of the 21st International Teletraffic Conference*, 2009.

[39] C. Raiciu, S. Barre, J. Iyengar, and B. Ford, “Architectural Guidelines for Multipath TCP Development, draft-ford-mptcp-architecture-00,” IETF, Draft, 2009.
[40] J. W. Suurballe, “Disjoint Paths in a Network,” *Networks*, 1974.
[41] G. Swallow, A. Atlas, and P. Pan, “Fast Reroute Extensions to RSVP-TE for LSP Tunnels,” IETF, RFC 4090, 2005.
[42] D. Topkis, “A k Shortest Path Algorithm for Adaptive Routing in Communications Networks,” *IEEE Transactions on Communications*, 1988.
[43] C. Villamizar, “OSPF Optimized Multipath (OSPF-OMP): draft-ietf-ospf-omp-02.txt,” Draft, 1999.
[44] S. N. Vutukury, “Multipath Routing Mechanisms for Traffic Engineering and Quality of Service in the Internet,” Ph.D. dissertation.
[45] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, “COPE: Traffic Engineering in Dynamic Networks,” in *SIGCOMM*, 2006.
[46] X. Yang and D. Wetherall, “Source Selectable Path Diversity via Routing Deflections,” in *SIGCOMM*, 2006.
[47] F. B. Zhan, “Three fastest shortest path algorithms on real road networks: Data structures and procedures,” http://publish.uwo.ca/~jmalczew/gida_1/Zhan/Zhan.htm.