Research Report

Low Complexity Link State Multipath Routing
The mDT algorithm

Pascal Mérindol, Jean-Jacques Pansiot, Stéphane Cateloin

RR-PM01-09
Mars 2009

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection,
UMR 7005, CNRS-UdS Pôle API, Bd Sébastien Brant BP 10413,
67412 ILLKIRCH CEDEX, FRANCE

Low Complexity Link State Multipath Routing

Pascal Mérindol
Université Catholique de Louvain (UCL)
Louvain la Neuve, Belgium
pascal.merindol@uclouvain.be

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

***Abstract*—Link state routing protocols such as OSPF or IS-IS currently use only best paths to forward IP packets throughout a domain. The optimality of sub-paths ensures consistency of hop by hop forwarding although paths, calculated using Dijkstra's algorithm, are recursively composed. According to the link metric, the diversity of existing paths can be underestimated using only best paths. Hence, it reduces potential benefits of multipath applications such as load balancing and fast rerouting. In this paper, we propose a low time complexity multipath computation algorithm able to calculate at least two paths with a different first hop between all pairs of nodes in the network if such next hops exist. Using real and generated topologies, we evaluate and compare the complexity of our proposition with several techniques. Simulation results suggest that the path diversity achieved with our proposition is approximatively the same that the one obtained using consecutive Dijsktra computations, but with a lower time complexity.**

I. Introduction

Routing is one of the key components of the Internet. Despite the potential benefits of multipath routing (e.g. [5] or [6]), most backbone networks still use unipath routing such as OSPF or IS-IS or their ECMP feature (Equal Cost MultiPath). With these routing protocols, the forwarding only changes upon topology variations and not upon traffic variations. Dynamic multipath routing (e.g. [16], [15], [8] or [3]) is able to provide several services such as load balancing, to reduce delays and improve throughput, and fast rerouting schemes in case of failures. The reliability of an IP network against failures and congestions depends on the reaction time necessary for the convergence of the underlying routing protocol. Proactive multiple paths calculation allows to accelerate this reaction time: pre-computed alternate paths can be directly used as backup paths without waiting for the routing protocol convergence. This proactive mechanism can improve the network response in case of troubles where such backup paths exist. To provide these functionalities, the set of forwarding alternatives has to be large enough to achieve a good path diversity. However, current routers only support ECMP. This feature corresponds to a simple variant of Dijkstra where equal cost paths are inherited along the shortest path tree (SPT). The optimality condition of sub-paths computed with ECMP restricts the number of loopfree paths and so reduces potential advantages of multipath routing.

In order to use multiple unequal cost paths between a pair of ingress and egress routers, there are two forwarding possibilities. On the one hand, source multipath forwarding schemes can use MPLS with a path signaling protocol (such as *RSVP-TE* [4]) to establish any desired paths. With this kind of approach, either the deployment is generalized in the whole network and does not scale very well (proportional to the square of the number of routers), either the reaction time can be as long as the notification delay on the return path.

On the other hand, multipath routing protocols with hop by hop forwarding needs to validate a set of next hops such that the recursive composition between neighbor routers does not create forwarding loops (see [14], [15] and [17]). The first limitation is the complexity in time, space and the number of messages exchanged to compute and validate loopfree paths.

In this paper, we propose a simple hop by hop scheme that does not require a signaling protocol to validate loopfree paths. If the validation procedure, whose goal is to verify the absence of loops, is local (without exchanging any message) and does not involve all routers, then the deployment can be incremental. Our approach is equivalent to ECMP in terms of time, space and message exchange complexity but allows to compute a greater diversity of forwarding alternatives.

In this paper, we propose the following contributions:

- a new graph decomposition analysis.
- two variants of the Dijkstra algorithm: Dijkstra-Transverse (DT) and multi-Dijkstra-Transverse (mDT).
- a proof that they compute at least two distinct next hops from the calculating node towards each node of the graph if such next hops exist.
- an evaluation of the efficiency and the complexity of our proposition compared to existing techniques.

This paper is organized as follows. Section II summarizes basic multipath routing notions and related work. Section III introduces our algorithms and their properties. Section IV presents our simulation results to underline the relevance and the low time complexity of our proposition.

II. Notations and context

Table I lists the graph definitions used in the paper. Notations are related to the multipath hop by hop forwarding context: computed paths are loopfree and first hop distinct. We order paths according to an additive metric C, and we focus on the best paths having distinct first hops. To distinguish equal cost paths, we consider the lexicographical order of first hops. For simplicity reasons we do not consider the multigraph issue: a first hop is equivalent to a successor node, the next hop. The valuation w denotes the weight of each directed link

TABLE I
NOTATIONS

Notations	Definitions
$G(N, E, w)$	oriented graph G with a set of nodes N, a set of edges E and a strictly positive valuation of edges w.
$e = (e.x, e.y)$	edge $e \in E$ connecting node x to node y we assume that $e^{-1} = (e.y, e.x) \in E$.
$k^-(x)$, $k^+(x)$	incoming and outgoing degrees of node x.
$succ(x)$	set of neighbors of node x ($\|succ(x)\| = k^+(x)$).
$P_j(s,d) = (e_1, ..., e_m)$	j^{th} best loopfree path linking s to d. Recursively, this is the best path whose first edge is distinct from the first edge of the $j-1$ best paths.
$C_j(s,d) = \sum_{i=1}^{m} w(e_i)$	cost of the path $P_j(s,d)$ $1 \leq j \leq k^+(s),\ 0 < m < \|N\|$.
$NH_j(s,d)$	j^{th} best next hop computed on s towards d. This is the first hop $e_1.y$ of $P_j(s,d)$.

used by the routing protocol. Let us define a safety property for distributed routing policies.

Definition: Loopfree routing property at the router level. *A multipath routing protocol is loopfree if it always converges to a stable state such that when any router s forwards a packet to any next hop v towards any destination d, this packet never comes back to s.*

With hop by hop link state multipath routing using multiple unequal cost paths, two phases may be necessary to ensure loopfree routing: a path computation algorithm and a validation process. We do not consider validation processes using a signaling protocol (such as it can be done with distance vector routing messages, see [15] for example).

With unipath or ECMP routing, the sub-path optimality condition guarantees the correctness of next hop composition. To increase the number of valid alternatives, the simplest rule to select a next hop v on a router s (such that $v \in succ(s)$) is the *downstream criteria* which can be expressed as follows:

$$C_1(v,d) < C_1(s,d) \tag{1}$$

This rule is referenced in the IS-IS standard ISO 8473, is used in OSPF-OMP [14] and is denoted LFI in [15] (with the particularity of avoiding routing loops even in transient periods of topology changes). This rule is called *one hop vision* in [17] where Yang and Wetherall introduce a set of rules whose flexibility allows to increase the number of valid neighbors thanks to a *two hops vision*. This set of rules is more complex: the forwarding mechanism is specific to the incoming interface and allows forwarding loops at the router level but not at the link level. Thus, a packet is never forwarded through the same link but it can enter the same router twice.

Authors suggests that minimizing the queue level should be the primary goal, however delays can increase if paths contain several times the same router and this unnecessarily consumes more resources (routers CPU, links bandwidth,...). We consider that the queue usage is not the only resource to save.

In order to perform loopfree routing, the validation process needs to compute a set of candidate next hops. A candidate next hop is a first hop of a computed path which is not yet validated for loopfree routing. On a given calculating node (a root node s), the simplest way to obtain an exhaustive candidate set is to compute the SPT of all neighbor nodes. Thus, router s can use the best costs information of its neighborhood. This approach is denoted **kD** in the following, and our analysis uses this technique as a reference. The complexity of kD depends on the number of neighbors: $k^+(s)+1$ instances of the Dijkstra algorithm are necessary to compute the local and neighborhood best costs. If a router has a large number of interfaces, the computation time can be too long. Even if this calculation is typically done offline, when a congestion or a failure occurs during this period, the router is unable to perform the traffic switching.

Another way is to use an enhanced SPT algorithm to locally compute multiple paths for each destination. For example, algorithms and implementations presented in [12] are designed to compute the set of K-shortest loopfree paths, but do not guarantee that these paths are first hop distinct. The K-shortest loopfree paths problem is not suited for simple hop by hop forwarding. Indeed, in order to forward packets via these K explicit paths, a signaling protocol is necessary to mark routes from the ingress router towards each egress router. Here we focus on distinct first hops computation ($K \leq k^+(s)$), and paths are implicity stored as candidate next hops. The objective of our approach is to compute a set of loopfree first hop disjoint paths with a lower complexity than **kD**. For this purpose, we calculate a set of costs $\{C_j(s,d)\}_{\ \forall d \in N}$ containing at least two entries for each destination node d in the graph. With an enhanced SPT algorithm able to compute such a set, rule (1) becomes:

$$C_j(s,d) - w(s,v) < C_1(s,d) \tag{2}$$

If $v = NH_j(s,d)$ satisfies rule (2), then (s,v) is a valid next hop. Thus, the j^{th} next hop v can be used by s to reach d and it satisfies the loopfree routing property at the router level. Note that: $\forall d \in N, C_j(s,d) - w(s,v) \geq C_1(v,d)$.

To sum up, our approach follows these three steps:

1) it uses an unmodified link state routing protocol such as OSPF or IS-IS to obtain topological information,
2) it uses a multipath computation algorithm (see section III) instead of Dijkstra to compute candidate next hops,
3) it uses condition (2) to select valid next hops.

III. CANDIDATE NEXT HOPS COMPUTATION

This section describes our path computation algorithms and an original edge partition analysis. Given a root node s, the set of edges of a graph can be partitioned into four subsets (we consider both directions of each edge):

- Edges corresponding to first hops of primary paths.
- Edges belonging to sub-trees corresponding to *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.

TABLE II
MULTIPATH TERMINOLOGY

Terms	Definitions
$branch_h(s)$	subtree of the SPT rooted at a neighbor h of 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 $branch_h(s)$
internal edge	an edge e is internal if it connects two nodes $e.x$ and $e.y$ belonging to a given $branch_h(s)$ and such that $e \notin branch_h(s)$
***k-transverse* path**	a path is k-transverse if it contains exactly k transverse edges and no internal edge
Simple *transverse* path $\mathcal{P} \in Pt(s,d)$	a 1-transverse path $(e_1,...,e_m)$ such that $(e_1,...,e_{m-1}) = P_1(s, e_{m-1}.y)$ and e_m is a transverse edge $(e_m.y = d)$
Backward *transverse* path $\mathcal{P} \in Pbt(s,d)$	a 1-transverse path $(e_1,...,e_m)$ such that for a z $_{(1\leq z<m)}$, $(e_1,...,e_z) \in Pt(s, e_z.y)$ and $(e_m^{-1},...,e_{z+1}^{-1}) = P_1(d, e_{z+1}.y)$
Forward *transverse* path $\mathcal{P} \in Pft(s,d)$	a 1-transverse path $(e_1,...,e_m)$ such that for a z, $(e_1,...,e_z) \in Pt(s,e_z.y) \vee Pbt(s,e_z.y)$ and $(e_{z+1},...,e_m) = P_1(e_{z+1}.x, d)$

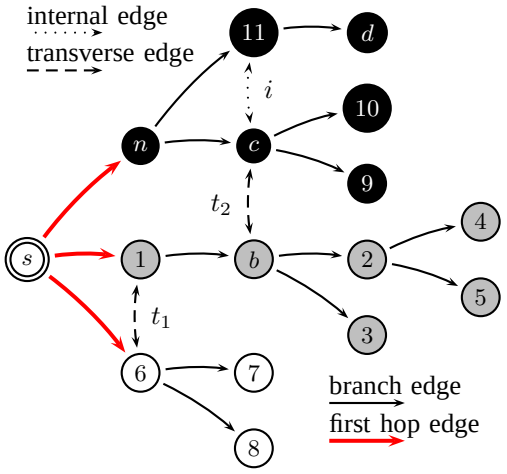

Fig. 1. Edge partition example

These four subsets exhaustively describe E because the set of branches contains all nodes (except the root node s) in the graph. Fig. 1 illustrates an edge partition on a simple graph (some nodes are identified with a letter to facilitate the reading of section III-B). In this graph (we consider w as a constant function), there are three branches (black, gray and white nodes), two *transverse* edges (dashed arcs denoted t_1 and t_2) and one *internal* edge (dotted arc denoted i). Edges (s,n), $(s,1)$ and $(s,6)$ correspond to the three first hops (red arcs) linking s to the three branches.

With multipath hop by hop routing, the *primary* path denotes the optimal path depending on a given metric and a lexicographic order to rank equal cost paths. Thus, for a given pair (s,d), an *alternate* path is a path whose first edge is distinct from the first one of the primary path $P_1(s,d)$. More generally, if the forwarding mechanism is distributed such as with hop by hop routing, then all alternate paths are first hop distinct. Table II summarizes all definitions related to transverse paths terminology. The path $((s,1),(1,b),(b,c))$ is *simple transverse* and the path $((s,1),(1,b),(b,c),(c,n))$ is *backward transverse*. Paths $\mathcal{P} = ((s,1),(1,b),(b,c),(c,n),(n,11),(11,d))$ and $\mathcal{P}' = ((s,6),(6,1),(1,b))$ are both *forward transverse*. However, $\mathcal{P}$ contains a sub path $((s,1),(1,b),(b,c),(c,n)) \in Pbt(s,n)$ whereas $\mathcal{P}'$ contains a sub path $((s,6),(6,1)) \in Pt(s,1)$. The path $((s,6),(6,1),(1,b),(b,c))$ is 2-transverse. The routing information base cannot directly use the set of candidate next hops corresponding to the first hops of 1-*transverse* path to perform forwarding, since routing loops may occur. Our approach needs a validation mechanism to select valid next hops among candidate next hops in order to guarantee the safety of forwarding. In this paper, we consider the rule (2) introduced in section II to validate candidate next hops. Due to space limitations, we do not discuss and evaluate rules allowing to use a higher route diversity (see [11]).

A. DT and mDT algorithms

In [10], we have proposed and described the Dijkstra-Transverse algorithm (**DT**). Here, we focus on DT properties that we have not presented in [10] (see section III-B) and on a DT improvement that we call multi-DT (**mDT**). However, the basics of DT and mDT are similar.

To sum up, DT and mDT compute a multipath cost matrix on a given root node (denoted s in the following). A multipath cost matrix contains an overestimation of best costs for all $(|N|-1)$ destinations and via all possible $(k^+(s))$ neighbors of s. The goal of these algorithms is to calculate a set of candidate next hops corresponding to costs associated to each neighbor. The calculation consists in two main stages:

1- Compute the best path tree and *transverse* edges.
2- Compute *backward* and *forward transverse* paths.

At each iteration, our algorithms compute the best 1-transverse paths depending on the first hop. Without an optimized structure to implement the best costs vector, the complexity of DT for each calculating node s is in the worst case:

$$O(|N|^2 + |E| + |N| \times k^+(s)) = O(|N|^2)$$

DT adds a time complexity proportional to the outgoing degree of the given root node s compared to Dijkstra. With a Fibonacci heap [7] to implement the best costs vector[1], it is possible to reduce the time complexity to:

$$O(|N|log_2|N| + |E| + |N| \times k^+(s))$$

The set of candidate next hops computed with DT does not always include all next hops corresponding to equal best cost paths. mDT (see algorithm 1) is able to solve this problem. With mDT, only the first computation phase of DT is modified by using a next hop matrix denoted Tp. This matrix represents the existence of a next hop per neighbor for each destination. Tp is updated at each edge exploration. Candidate next hops recording follows a transitive rule: $Tp(k,y) \leftarrow Tp(k,x)$ with $y \in succ(x)$, $k \in succ(s)$. Initially, if $x = s$ then

[1]The minimum extraction has an unitary cost whereas the minimum suppression has an amortized cost in $O(log_2(|N|))$. For simplicity reasons, evaluations results that we present in this paper only rely on array lists.

$Tp(y,y) \leftarrow y$. With ECMP, the update of Tp is performed only if $Tc(x) + w(x,y) \leq Tc(y)$. We have chosen to generalize this approach to improve the upper bound on the cost of *forward transverse* paths composed with a *backward transverse* path. This generalization increases the number of validated next hops. Indeed, during the exploration of the set of successors of node x, if node y is not already marked, it inherits all forwarding alternatives of x, including when (x,y) is an internal edge. In this case, the next hop inheritance is not restricted to branches as with DT: y is not the *son* of x on a primary path. mDT allows to use all forwarding alternatives already computed towards x. This set of paths is not limited to 1-transverse alternatives, it can contain alternate paths with several internal or transverse edges. The mDT computation is based on the order of node exploration which depends on the rank of costs stored in Tc. With mDT, the first computation phase is able to calculate all candidate next hops corresponding to ECMP alternatives. Recursively, the cost inheritance takes into account all the sets of equal best cost paths for all marked nodes. The complexity of mDT is slightly greater than the one of DT: for each iteration of the main loop, $k^+(s)$ operations are necessary to execute the inheritance of next hops and their costs. The worst case complexity of mDT is in $O(|N|^2 + E \times k^+(s))$ without an optimized structure for Tc.

B. Properties of DT and mDT

In this section, we prove the ability of our algorithms to compute at least two candidate next hops between each pair of nodes in the graph if such next hops exist.

Property 1. *DT computes all* 1*-transverse paths, and mDT computes all paths computed with DT and all equal best cost paths.*

The proof of these properties relies on next hops inheritance performed by DT and mDT (for more details, see [10]).
Now, let us define a major property of 1-transverse paths.

Property 2. *If there exists an alternate path $P(s,d)$, then there exists a* 1*-transverse path between s and d.*

The demonstration of this property relies on two lemmas.

Lemma 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 of Lemma 1: Let $\mathcal{P} = (e_1, e_2, ..., e_i, ..., e_m)$ be an alternate path from s to d where $e_i = (x,y)$ is the last transverse edge of $\mathcal{P}$ and consider $P_1(s,x)$ the shortest path from s to x. Then either $P_1(s,x)$ is empty because $x = s$ and $i = 1$, or $P_1(s,x)$ is a primary path which is not longer than $(e_1, e_2, ..., e_{i-1})$. Let $\circ$ be the operator representing the path concatenation. In both cases, there exists a path P' such that $\mathcal{P}' = P_1(s,x) \circ (e_i, ..., e_m)$ is an alternate path with only one transverse edge and which is not longer than $\mathcal{P}$. ■

Figure 1 illustrates lemma 1. The 2-transverse path $\mathcal{P} = ((s,6),(6,1),(1,b),(b,c))$ between s and c via the neighbor node 6 uses $branch_1(s)$ to reach the transverse edge (b,c).

Algorithm 1 multi-Dijkstra-Transverse algorithm

procedure MULTI-DT ($G(N,E,w), s$)
  $Mc_{k^+(s),|N|-1}$: Cost matrix
  $Tp_{k^+(s),|N|-1}$: Next hop matrix
  $Tc_{|N|-1}$: List of best costs
  $F_{|N|-1}$: List of father nodes
  $T_{|N|-1}$: List of marked nodes
  $Mc(k,d)$,$Tp(k,d)$ and $Tc(d) \leftarrow \infty$, $\forall d \in N,\ k \in succ(s)$
  $Tc(s) \leftarrow 0$
  ▷ *SPT and transverse path computation*
  while $|T| < |N|$ **do**
    Choose the node x ($x \notin T$) of minimum cost $Tc(x)$
    for $y \in succ(x)$ **do**
      for $k \in succ(s) | Tp(k,x) \neq \infty$ **do**
        Update $Tp(k,y)$
        if $Mc(Tp(k,x),x) + w(x,y) < Mc(Tp(k,y),y)$ **then**
          Update $Mc(Tp(k,y),y)$
        end if
      end for
      if $Tc(x) + w(x,y) < Tc(y)$ **then**
        Update $Tc(y)$, $F_s(y) = x$
      end if
    end for
    Put x in T
  end while
  ▷ *Backward and forward composition*
  for $i : |N| \rightarrow 1$ **do**
    for $y \in succ(s)$ **do**
      if $Mc(y,T(i)) + w(T(i),F(T(i))) < Mc(y,F(T(i)))$ **then**
        Update $Mc(y,F(T(i)))$
      end if
    end for
  end for
  for $i : 1 \rightarrow |N|$ **do**
    for $y \in succ(s)$ **do**
      if $Mc(y,F(T(i))) + w(F(T(i)),T(i)) < Mc(y,T(i))$ **then**
        Update $Mc(y,T(i))$
      end if
    end for
  end for
  Return Mc
end procedure

There exists an alternate simple transverse path $\mathcal{P}' = P_1(s,b) \circ ((b,c))$. Note that the existence of a path P with several transverse edges implies that DT (and mDT) implicitly records a 1-transverse path $\mathcal{P}'$ in the cost matrix Mc with a cost lower or equal to the cost of P.

Lemma 2. *If there exists an alternate path from s to d with one transverse edge, then there exists a 1-transverse path linking s and d.*

Proof of Lemma 2: Let $\mathcal{P} = (e_1, ..., e_i, ..., e_m)$ be such an alternate path where $e_i = (b,c)$ is the unique transverse edge. Without loss of generality we may assume that $P_1(s,b) = (e_1, ..., e_{i-1})$ is a primary path (see lemma1) without any internal edge. Note that $(e_1, ..., e_i) \in Pt(s,c)$. To characterize the differences between transverse paths, we use an "ancestor function". An ancestor a of a node x is a node such that there exists a primary path $P_1(a,x)$ included in the SPT rooted at s. The closest common ancestor n of nodes x and y is an ancestor of x and y such that for any common ancestor a of x and y, a is also an ancestor of n.
Let n be the closest common ancestor of nodes c and d.

1) If $n = c$ then there exists a forward transverse path linking s and d: a simple transverse path between s and c and a primary path between c and d.
2) Else if $n = d$ then there exists a backward transverse path linking s and d: a simple transverse path between s and c and a path in the reverse direction of the primary path between d and c [2].
3) Else if $n \neq c, n \neq d$, then n is the node where the branch including d and c is subdivided into two sub-branches, one containing c, the other containing d [3]. In this case, there exists a forward transverse path linking s and d which contains a backward transverse path $\in Pbt(s,n)$ and a primary path $P_1(n,d)$.

Thus, in each case, the existence of a *1-transverse* path allowing to reach d is verified. ■

Figure 1 illustrates lemma 2. Although the alternate path $((s,1),(1,b),(b,c),(c,11),(11,d))$ is not 1-transverse because it contains an internal edge $(c,11)$, there exists a forward transverse path $((s,1),(1,b),(b,c),(c,n),(n,11),(11,d))$. In this case, the internal edge i is bypassed with a backward composition followed by a forward composition. It allows to compute the alternate next hop 1 to reach d.
Thanks to the backward and forward composition, if there exists a 1-transverse path, then DT finds it. These two phases allow to use edges of the SPT in both directions. Moreover, DT considers all transverse edges because, as it is the case for the classical Dijkstra algorithm, all edges must be explored in order to mark all nodes. The difference is that DT implicitly stores longer or equal cost paths in the cost matrix.

Corollary 1. *For any pair of nodes (s,d), if there exists an alternate path from s to d, then DT and mDT allow s to compute at least two candidate next hops towards d.*

[2]We assume that $e \in E \Rightarrow e^{-1} \in E$.
[3]Note that we know that $C_1(s,c) > C_1(s,n)$ and $C_1(s,d) > C_1(s,n)$.

Corollary 2. *If the graph contains no bridge edge, then DT and mDT allow s to compute at least two candidate next hops between any node and any other node of the graph.*

For a given destination, the corollary 1 allows to conclude that the number of candidate next hops is at least 2 if there exists an alternate path linking s and d. Corollary 2 is more specific, if the network is 2-edge connected, then corollary 1 can be extended for all pairs of routers.

IV. Evaluation

We use the Network Simulator 2 (ns2, [2]) to compare several routing approaches. ECMP is already implemented within the link state module of ns2. We have extended ns2 to implement DT, mDT, kD and the downstream criteria, rule (2), in the routing module (see [1] to find the implementation).

A. Topologies and simulations setup

We present results obtained on three different kinds of topologies. The first category of networks are real topologies with actual IGP weights (for confidentiality, we approximate their size in Table III). Topologies denoted ISP1 and ISP2 are commercial networks covering an European country. ISP3 and ISP4 are Tier-1 ISP networks. The second category of topologies were chosen among the Rocketfuel inferred set of maps given in [9].
We have also used the Igen topology generator ([13]) in order to obtain a set of evaluation topologies of various sizes. We have generated 10 topologies containing between 20 and 200 nodes using the K-medoid parameter, the delay-triangulation heuristic and a 2-sprint pop design. The K parameter that determine the number of routers per cluster is chosen such that $K = \frac{|N|}{10}$, so that each cluster contains approximatively 10 routers for each generated topology. These parameters offer a great physical diversity to measure the relevance of our proposition to achieve the same level of diversity as computed with kD. The link valuation used for this third category is the inverse of the link capacity. The mean degree, denoted k, is approximatively the same for each generated topology: $k \sim 4$. These networks represent access backbones and contain two kinds of links: 155Mbps for access links and 10Gbps for backbone links (so that weights of links are respectively 64 and 1).

B. Results

C. Diversity results

First, we have measured the path diversity (see Fig.2). We have calculated the total number of candidate next hops obtained with ECMP (denoted EC), DT, mDT, and multiple Dijkstra computations (kD). Results are represented as a performance ratio between the considered technique and kD for all routers of a given network. kD provides the best diversity but with a higher computation cost. We observe that DT and mDT are able to compute approximatively 90% of candidate next hops obtained with kD, while ECMP obtains a performance ratio only between 60% and 80%.

TABLE III
EVALUATION RESULTS ON REAL AND INFERRED TOPOLOGIES

Network name	Size		Candidate next hops				Validated next hops				Number of operations			
			mean	ratio/kD (%)			mean	ratio/kD (%)			mean	ratio/kD (%)		
	$\|N\|$	$\|E\|$	kD	EC	DT	mDT	kD	EC	DT	mDT	kD	EC	DT	mDT
ISP1	25	50	1.46	76	97	97	1.10	97	100	100	489	60	66	75
ISP2	50	200	3.58	43	93	97	1.79	69	89	94	6730	30	32	32.5
ISP3	110	350	2.70	55	89	92	1.45	82	97	99	8079	38	41	43.5
ISP4	210	880	3.73	44	86	88	1.81	72	96	99	41747	27	28	31
Exodus	79	294	3.58	44	88	96	1.73	58	94	99	5569	29	34	37
Ebone	87	322	3.49	46	90	96	1.76	77	93	99	9698	30	33	36
Telstra	104	304	2.30	72	92	95	1.30	90	98	99	6526	54	57	59
Above	141	748	5.29	34	86	97	2.50	58	89	99	40143	18.5	20	23
Tiscali	161	656	3.68	54	91	97	1.97	74	92	97	31044	27	29	32

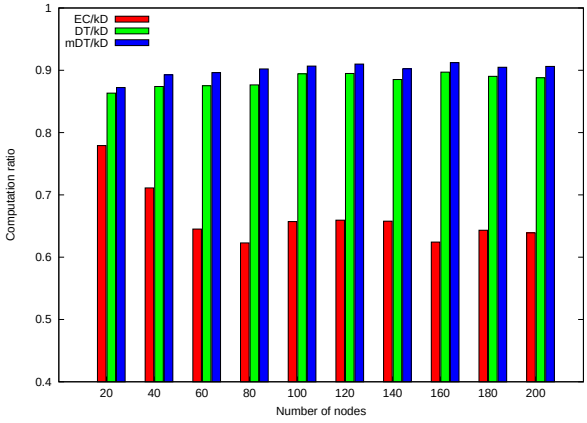

Fig. 2. Number of candidate next hops (Igen topologies)

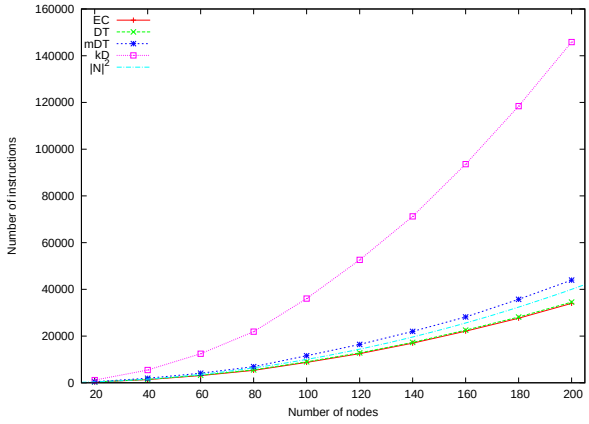

Fig. 3. Number of operations (Igen topologies)

D. Complexity results

Then, we have compared the time complexities of the fore mentionned algorithms (see Fig. 3). We have represented the execution time measured in number of operations needed by DT, mDT and kD to compute their set of candidate next hops. The number of operations is an average computed for each router. This value takes into account all operations necessary to extract the min of Tc and perform update of Tc, Mc and Tp. We notice that the time saved with DT or mDT is really significant compared to kD. The number of operations needed by kD is approximatively $k \times |N|^2$ whereas mDT and DT need approximatively $|N|^2$ operations. This complexity is equivalent to the worst case of an ECMP computation. The time complexity upper bound is reached because some routers of Igen topologies have a high degree of connectivity.

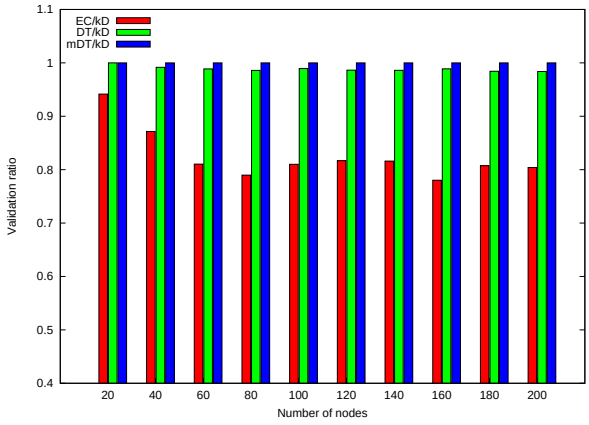

Fig. 4. Number of validated next hops (Igen topologies)

E. Loopfree diversity results

Finally, we have compared the number of validated next hops that are selected with the downstream criteria (rule 2) depending on the computation algorithm (see Fig. 4). We remark that mDT allows to validate as many next hops as kD. This result can be explained by the specific valuation function w of our set of generated topologies: there are only two very distant weights used in these networks.

F. General results and discussion

Results given in Table III illustrate the same evaluation of performance ratios and complexity on the set of real and inferred topologies. For these sets of topologies, Table III also shows candidate and valid next hops average per destination

obtained with kD. Diversity ratio results are similar to the ones obtained with Igen although degrees and weights distributions are completely different. The main difference comes from the time complexity evaluation. On these topologies, the maximum degree of nodes is two times lower than with Igen topologies. The measured complexity is far away from the theoretical worst case. More generally, several parameters, such as the valuation function w or the degree distribution may strongly influence complexity measures, and thus the performance of algorithms. For example, if w is a constant function, rule (2) is equivalent to ECMP. Thus, in this case, the number of valid next hops is the same for mDT, kD and ECMP. Another key point is the fact that the alternate paths which are not computed with mDT have a cost generally much more greater than the one of the primary path, that is why the ratio of loopfree alternatives between mDT and kD is close to 100%.

To summarize, although DT and mDT consume less processor resources than kD, they are able to offer almost the same diversity in terms of validated next hops.

V. Conclusion

Multipath routing enhances the network reachability and allows load balancing to circumvent congestions or failures. However, the overhead imposed by signaling messages, the time and space complexity can hamper its deployment. In this paper, we propose a simple scheme that is able to generate a greater path diversity than ECMP with an equivalent overhead. Our path computation algorithms, Dijkstra-Transverse, and its improvement multi-DT, allow to compute at least two candidate next hops between all pairs of routers if such next hops exist. To validate candidate next hops in a distributed manner, we have considered the simplest loopfree routing rule, the downstream criteria. Our evaluations suggest that the gain of time is very significant. We show that the number of next hops validated with the downstream criteria is slightly the same using mDT or a Dijkstra computation per neighbor. Moreover, our proposition can be integrated in OSPF or IS-IS by replacing the path computation algorithm without any change in the protocol. It can be deployed incrementally, some routers using ECMP and others DT or mDT. Our proposition can be extended to compute backup next hops only selected if a failure occurs.

Acknowledgement

The research results presented herein have received support from Trilogy (http://www.trilogy-project.eu), a research project (ICT-216372) partially funded by the European Community under its Seventh Framework Programme. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document. The authors would like to gratefully acknowledge Pierre Francois and Olivier Bonaventure for their comments.

References

[1] "Implementation of dt and mdt in ns2," http://www-r2.u-strasbg.fr/~merindol/uploads/Research/DT.tar.gz.
[2] "The network simulator- ns2," http://www.isi.edu/nsnam/ns.
[3] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs," in *SIGCOMM*, 2003.
[4] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE : Extensions to RSVP for lsp tunnels," RFC 3209, 2001.
[5] R. Banner and A. Orda, "Multipath routing algorithms for congestion minimization," *IEEE/ACM Trans. Netw.*, 2007.
[6] I. Cidon, R. Rom, and Y. Shavitt, "Analysis of multi-path routing," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, 1999.
[7] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
[8] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," in *SIGCOMM*, 2005.
[9] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *ACM SIGCOMM Internet Measurement Workshop*, 2002.
[10] P. Mérindol, J.-J. Pansiot, and S. Cateloin, "Path computation for incoming interface multipath routing," in *ECUMN*, 2007.
[11] P. Mérindol, J.-J. Pansiot, and S. Cateloin, "Improving load balancing with multipath routing," in *ICCCN*, 2008.
[12] M. Pascoal, "Implementations and empirical comparison for k shortest loopless path algorithms," in *The Ninth DIMACS Implementation Challenge: The Shortest Path Problem*, 2006.
[13] B. Quoitin, "Topology generation through network design heuristics," http://www.info.ucl.ac.be/~bqu/igen/.
[14] C. Villamizar, "Ospf optimized multipath (ospf-omp): draft-ietf-ospf-omp-02.txt," IETF, Draft, Feb. 1999.
[15] S. N. Vutukury, "Multipath routing mechanisms for traffic engineering and quality of service in the internet," Ph.D. dissertation, 2001.
[16] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: traffic engineering in dynamic networks," in *SIGCOMM*, 2006.
[17] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *SIGCOMM*, vol. 36, 2006.