Improving Load Balancing with Multipath Routing

Pascal Mérindol, Jean-Jacques Pansiot, Stéphane Cateloin
LSIIT - ULP - CNRS
Pôle API Boulevard Sébastien Brant
67400 ILLKIRCH FRANCE
Email:{merindol,pansiot,cateloin}@dpt-info.u-strasbg.fr

***Abstract*— Internet service providers have to provision network resources to optimize bandwidth utilization. Dynamic routing protocols take traffic variations into account to control the load distribution. Multipath routing protocols attempt to take advantage of the path diversity to bring network robustness and reliability. Indeed, with a specific traffic engineering policy, they enable load balancing across several paths. Our aim is to compute a set of loopfree paths in order to allow routers to share the load on several next hops depending on current load measurement. In this paper, we first describe our original Incoming Interface Multipath Routing technique, DT(p), then we present a scheme for load balancing, DT(p)-TE, based on link monitoring. We evaluate and compare our technique with several existing approaches by a set of simulations, using different scenarios and topologies.**

I. Introduction

Traffic engineering (TE) is a critical issue in large IP backbones. Dynamic routing is able to circumvent congested links in order to improve the quality of applications such as HD video and VoIP services. However, overhead imposed by the frequency of link-state updates, new path activation, and signaling hampers its deployment. Multipath routing can balance network load to improve the streaming quality of long lived flows. Short flows are not specifically concerned with dynamic routing because their duration can be smaller than the link state updating period (the duration between two consecutive messages of traffic measurement). Despite the potential benefits in terms of resource control, most backbone networks still use static routing (OSPF [13] or IS-IS [15] consider the topology changes but not residual bandwidth fluctuations) because dynamic routing can lead to route flapping, strong traffic oscillations and excessive signaling messages overhead. Packets routed on outdated information can lead to serious load oscillations if the system does not react quickly enough. TE objectives can be obtained by routing traffic demands on multiple paths. Our approach to reach an efficient multipath routing is divided into four tasks:

a) Compute and position paths.
b) Analyze local traffic activities and advertise the availability of local resources in the network.
c) Define load balancing policy depending on the computed (and received) information.
d) Split the traffic among routes.

This paper deals with these four objectives without considering the possibility to inform upstream nodes about local residual resources. Multipath routing is also an interesting tool to provide a fast reaction to protect networks from link or node failure. However, conditions to use backup paths have to be stricter than with unipath routing protocols because alternate routes can be used even without failure.
We propose the following contributions:

a) We introduce the *Dijkstra Transverse* algorithm (DT) and DT(p) our path validation at depth *p* procedure.
b) Then, we analyze the load balancing issue in case of congestion. We present DT(p)-TE, a mechanism which locally analyzes the bandwidth utilization to dynamically compute sharing proportions.
c) We finally evaluate DT(p), with DT(p)-TE as a load balancer scheme, using simulations with several topologies and traffic traces.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces our incoming interface multipath routing technique and a simple mechanism to balance the load on computed routes. Section 4 presents our simulation scenarios and our main results to underline the relevance of our techniques.

II. Related work

There are two main families of routing protocols enabling load balancing on multiple paths. The first family uses labeling or source routing mechanisms (deployed above traditional IP-routing) whereas the second family makes use of hop by hop routing protocols.
Source routing multipath techniques generally contain two stages for path provisioning (tunnels built as an overlay network above the link layer):

a) Source path computation algorithms (such as *K's* best path [7] or *CRA* [14]) to reduce delays, improve throughput or compute efficient protection paths.
b) Path signaling protocols (such as *RSVP-TE* [5], *CR-LDP* [8]) to position computed paths.

The main advantage of this type of technique is the ease with which the administrator can choose TE tunnels, without considering loop presence as in distributed methods. For protection and restoration aspects for example, it is important that the bypass tunnel guarantees the bandwidth requirements assumed by the primary path. Source path computation allows to easily verify this kind of constraint. Recent TE protocols map the traffic of an Ingress-Egress routers pair onto multiple routes and adapt the load of each route depending on real time

measurements, in order to avoid hot spots and cope with failure events. Protocols like TeXCP [9] and MATE [6] use probing packets to measure network response time and thus increase the network robustness. However, with path protection or load balancing objectives, only ingress nodes which label or reserve path resources until the egress nodes are able to shift the traffic from one path to another. The reaction time can be therefore as long as the notification delay on the return path. With link by link protection (for failure or congestion avoidance), the reaction can be faster, but does not scale very well, since the number of bypass tunnels can quickly become too large and similarly with the signaling messages number. Consequently, the extensibility in terms of Ingress-Egress routers pairs using such techniques is limited. In an *MPLS* cloud, only border routers can play this role in a reasonable perspective when local path diversity is a key issue for TE requirements.
The second family gathers IP load balancing methods which can partially solve these problems. However, they have to guarantee that IP packets in transit will not loop. We therefore have to define the properties of loop free routing for distributed policies. We distinguish between *local traffic* coming from the router itself or from directly attached subnetworks, and *transit traffic* coming from other routers. For simplicity, we assume that they come from distinct interfaces. In figure 1, we illustrate the difference between transit traffic, coming from an upstream router *p* for example, and local traffic, potentially coming from *s* or its subnetwork. This figure can serve as a basis for definitions and conditions described in this paragraph.

Definition 1 (Loopfree routing property at the node level): A multipath routing protocol is loopfree if, whenever a router *s* sends a local or transit packet to any next hop *v* towards a destination *d*, this packet never comes back to *s*.

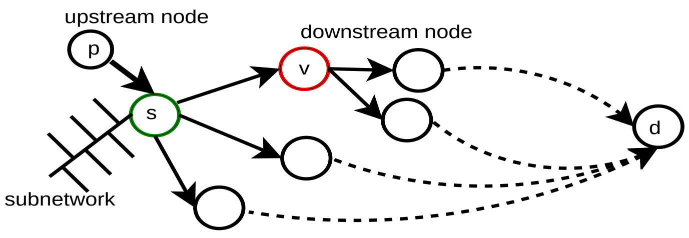

Fig. 1. Hop by hop loop free routing

Table I gives definitions required to express the loopfree routing property. We order paths depending on a given link state metric C, and we focus on the best paths whose first edges are distinct. To differentiate equal cost paths, we consider the lexicographical order of first hops. The valuation w denotes the weight of each link, it can be the inverse of the link capacity for example. Conditions given in Table II are sufficient but not necessary to define loopfree alternate paths. Consequently, according to the topology characteristics, an alternate path cannot always be found, even if there exists one.

Deployed routers usually implement the ECMP (Equal Cost MultiPath) feature of routing protocols such as OSPF or IS-IS.

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 w of edges.
|N|,|E|	respective cardinal of sets N and E.
$e = e.x, e.y$	edge e $\in$ E connecting node x to node y.
$k^-(x)$, $k^+(x)$	incoming and outgoing degree of node x.
$P_j(s,d) = \{e_1, ..., e_m\}$	j^{th} best 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.
$C_j(s,d) = \sum_{i=1}^{m} w(e_i)$	j^{th} best cost computed on s towards 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 $P_j(s,d)$.
$NH(s,p,d)$	set of next hops validated on the router s for the upstream router p as input and towards destination d.

TABLE I
NOTATIONS

Condition for v	Name	References
$C_j(s,d) = C_1(s,d) \wedge v = NH_j(s,d)$	ECMP	[13],[15]
$C_1(v,d) < C_1(s,d)$	LFI	[20],[21]
$C_1(v,d) < C_1(p,d)$	SPD	[21]

TABLE II
LOOP FREE RULES

Condition LFI ([20]) requires a signaling procedure to obtain the costs computed by each neighbor for each destination. The last condition SPD (Source Path Deflection) is used in [21] to avoid loop at the link level. This article presents a set of rules whose increasing flexibility allows to widen the space of valid neighbors. The loopfree routing property at the node level is not verified. Indeed, a packet can transit twice by the same router but never by the same link. Authors argue that the queue is the primary resource to save, however delays can increase if paths contain several times the same router and this consumes more resources. We do not think that the queue usage is the only resource that a network administrator has to take care of. Condition SPD needs also an enhanced Shortest Path First (SPF) algorithm to compute best path costs of the neighborhood.

III. INCOMING INTERFACE MULTIPATH ROUTING DT(P) AND LOAD BALANCING

This section first describes our path computation and validation algorithms. We present our enhanced SPF algorithm and introduce our validation protocol at depth p. These two stages produce loop free routes with low computational and signaling messages overhead. The ability of our proposition to benefit as much as possible from path diversity is due to the distinction done on the interface on which the packet arrives. Other works such as "Source Selectable Path Diversity via Routing Deflection" [21] and U-TURN [3] (an extension of the "Loop Free Alternates" technique [4] to reroute the traffic in case of failure) also use this idea.
Then the TE section deals with the load balancing issue. We briefly expose the problem of the load sharing and finally

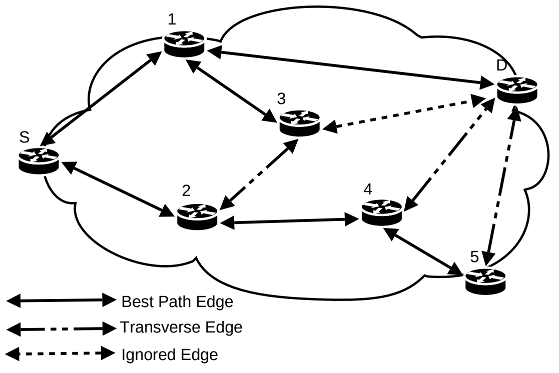

Fig. 2. Six routes can link *S* and *D* with DT(1)

propose a solution whose main advantage is simplicity.

A. Path computation and validation

In this section we present the ability of our DT(p) technique to compute a large set of paths. To illustrate the relevance of our method, we consider the simple network given in figure 2. If all links have the same weight, only the best path via router *1* can be used to link *S* and *D* with existing distributed multipath techniques (except for SPD, but the loopfree routing property at node level is not verified).
In this paper, we will show how, with our proposition, *S* can benefit from six paths to reach *D* without creating loops.

1) Dijkstra Transverse (DT): This paragraph gives a short description of our enhanced SPF algorithm to compute a multipath cost matrix. A cost matrix computed on *s* contains an estimated best cost for $|N|$ destinations and via all possible ($k^+(s)$) adjacent neighbors.

Terms	Definitions
branch $branch_h(s)$	all best paths $P_1(s,d)$ in the best path tree which have the same first edge $\{s,h\}$.
transverse	an edge is transverse if it connects two distinct branches.
simple transverse $Pt(s,d)$	a path of m edges $\{e_1,e_2,...,e_m\}$ such that $\{e_1,e_2,...,e_{m-1}\}$ forms a best path $P_1(s,e_{m-1}.y)$ and such that e_m is a transverse edge.
backward transverse $Pbt(s,d)$	a path of m edges $\{e_1,e_2,...,e_m\}$ such that for a z i.e $1<z<m$, $\{e_1,...,e_z\}$ is simple transverse, and such that $\{e_m,e_{m-1},...,e_z\}$ is a best path $P_1(d,e_z.y)$.
forward transverse $Pft(s,d)$	a path of m edges $\{e_1,e_2,...,e_m\}$ such that for a w i.e $1<z<m$, $\{e_1,...,e_z\}$ is either simple transverse or backward-transverse and such that, $\{e_m,e_{m-1},...,e_z\}$ is a best path $P_1(e_z.y,d)$.

TABLE III
TERMINOLOGY

DT computation consists in three main steps:
(definitions are given in Table III)

a) Compute the best path tree and *simple transverse* paths.
b) Construct a *backward transverse* path set and add it to the previous set.
c) Construct a *forward transverse* path set and add it to the previous set.

At each step, DT stores the best alternatives depending on the first hop, the outgoing interface. We must consider that edges in the best path tree are symmetric in valuation and existence (duplex links) for backward transverse path computation (step b). The complexity of our algorithm in each root node *s* is, in the worst case, and without an optimized structure to implement the best cost table:

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

Therefore, the DT algorithm introduces a slight overhead in calculation time (proportional to the outgoing degree: $k^+(s)$). A complete description of our algorithm is given in [12].
If we use these sets of paths computed on each router without considering the packet origin, this causes loops due to hop by hop routing. The composition of next hops computed with DT needs a validation protocol to transform these next hop candidates in valid loopfree next hops. We first describe a path validation with a *one hop vision*, DT(1): the path validity is checked in the one hop neighborhood at the incoming interface granularity.

2) DT(1): Since hop by hop routing with unequal cost paths may induce loops, we enter the neighbor-node validation phase. Initially, only equal best cost paths are valid without considering the incoming interface. Now routers have to exchange best path cost informations to validate other (simple, backward or forward) computed paths. A router *s* sends, for all destinations $d \in N$, a message to its downstream router $v = NH_i(s,d)$ which contains the best cost to reach *d*, $C_1(s,d)$. To validate a NH, we use the following condition:

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

If this condition is verified on a router *v*, for an upstream router *s* and towards a destination *d*, the loopfree routing property is guaranteed for the next hop $NH_j(v,d)$. For all *j* verifying (1), $NH_j(v,d)$ is marked as a valid next hop for traffic coming from *s* to *d*, and *v* validates a new routing row *(s,* $NH_j(v,d)$*, d)*. A routing row on a router *s* is a triple *(p, n, d)* meaning that traffic coming via *p* for *d* can be sent to next hop *n* where $n = NH_j(s,d) \in NH(p,s,d)$. If (1) is verified for at least one *j*, *v* sends a positive answer to *s*, and *s* marks *v* as a valid NH for local traffic towards *d*. Then, *s* is able to use $NH_i(s,d) = v$ for its local traffic. The main advantage of our technique is the use of a candidate routing table allowing to individually check the validity of each NH (corresponding to the first hop of a path P_j). Condition (1) means that the first hop $NH_j(v,d)$ of a path P_j is valid for the couple (s,d) and for lower cost paths. Indeed, if *v*, an adjacent node of *s*, guarantees a cost C equal to the best one that *s* has, for a given destination *d*, then, one hop further, the path cost is strictly less than C. Note that P_j is validated on *v* for *s* as input (transit traffic coming from *s*) only if *v* validates $NH_j(v,d)$ for its local traffic or if $C_j(v,d) = C_1(v,d)$. To increase the number of validated paths, we have to increase the depth of the validation process, this is DT(p). DT(p) prunes the DT subgraph at the depth *p* to avoid routing loops.

3) DT(p): In order to introduce DT(p), we have to define the notion of *route* as opposed to the notion of *path*. With distributed (hop by hop) routing, only the first hop of a path is actually used for routing.

Definition 2 (Route): Formally we denote a route of *m* hops linking a source *s* and a destination *d* : $R_m(s,d)$, and we denote $NH(s,s,d)$, the set of validated next hops of paths computed by DT on *s*, for its local traffic. Hence, a route $R_m(s,d)$ is a composition of validated NHs (depending on the incoming interface) and takes this form ($s = r_0$, $d = r_m$):

$$R_m(s,d) = \{r_1, r_2, \cdots, r_i, r_{i+1}, \cdots, r_m\}$$

with $r_{i+1} \in NH(r_i, r_{i-1}, d)$ and $r_1 \in NH(s,s,d)$.
With this terminology we can describe our breadth first search loop detection method with *p* nodes in depth. This is a wave of messages called *query* triggered on each downstream router $v = r_1$ where DT(1) does not succeed for the upstream node s on the k^{th} NH of r_1: $NH_k(r_1,d)$. These messages $query(s,d,c,q,P)$ contain $c = C_1(s,d)$, the best cost for s, *q* ($1 \leq q \leq p$) the number of remaining hops and *P* the set of tested routers. In the following we describe our algorithm for fixed *s* and *d*. The aim is to determine if a NH is valid, even if it does not satisfy condition (1). With *p>1*, DT(p) cannot benefit from the granularity of the incoming interface. If $p > 1$, condition (1) has to be verified for all NHs computed by DT. However, a router has only to take care of loops coming back to itself. A router r_θ ($0 < \theta < p$) can appear twice or more in the validation phase. The wave triggered on a router r_1 which does not belong to *NH(s,s,d)* with *p=1* (or if a router $r_2 = NH_k(r_1,d)$ does not belong to $NH(r_1,s,d)$), must explore, in a radius of *p-1*, all NH compositions to test the set paths generated by DT. If r_1, a neighbor of *s*, does not verify condition (1) on $NH_k(r_1,d) = r_2$, it forwards the validation message $query(s,d,c,q-1,r_1)$ to r_2 and waits for a reply. When a node r_{i+1} receives a $query(s,d,c,q,P)$ from r_i, the pseudo code of the DT(p) algorithm can take this form:

```
▷ if $NH_j(r_{i+1},d)$ satisfies (1), $r_{i+1}$ stores a VALID result for $NH_j(r_{i+1},d)$
▷ else if $NH_j(r_{i+1},d) = r_\theta$ with $r_\theta \in P = \{r_1,...,r_i\}$, $r_{i+1}$ stores a SKIP result for $NH_j(r_{i+1},d)$
▷ else if $NH_j(r_{i+1},d) = s$, $r_{i+1}$ replies with a LOOP result to $r_i$
▷ else if q > 0, $r_{i+1}$ sends a $query(s,d,c,q-1,P)$ with $P \leftarrow P \cup r_{i+1}$ to its candidate $NH_j(r_{i+1},d)$
▷ else the max depth p has been reached without success and a LOOP result is returned to $r_i$
```

When r_i, with $i > 1$, has a result/reply for all its candidate NHs it computes its own result which is the max of all responses (the order of replies/results verifies $LOOP > VALID > SKIP$) and sends it to r_{i-1}. $NH(r_{i+1}, r_i, d) \neq \emptyset$ only if r_i verifies condition (1) on a NH of r_{i+1}, or if r_{i+1} receives a VALID result coming from a router r_{i+2} for (r_i,d). If r_1 receives a VALID answer from r_2, it validates a routing row (s, r_2, d) and transmits this information to *s* (if there exists a routing row $(r_1, r_2, d) \Rightarrow r_2 \in NH(r_1, r_1, d)$).

With the DT(p) procedure, each router r_i in a route $R_m(s,d)$ guarantees the two following properties ($\forall i \in [0,m]$):

(a) If $\{k | NH_k(r_{i+1},d) = r_{i+2} \wedge C_k(r_{i+1},d) > C_1(r_i,d)\}$, routers $r_{i+q,\, 2 \leq q \leq p}$ guarantee in a maximum radius of $p-q$, for each possible NHs composition with DT (except SKIP NH), a cost less than $C_1(r_i,d)$.

(b) $NH(r_i, r_{i+1}, d) \subset NH(r_{i+1}, r_{i+1}, d)$.

Note for property (a) that, if $NH_k(r_{i+q},d) = r_{i+j}$, with $1 \leq j < q \leq p$, this generates a SKIP result for the k^{th} NH of r_{i+q}: $NH_k(r_{i+q},d)$. Routers $r_{i+q,\, q \leq p}$ must be distinct from r_i (LOOP result), so property (a) permits to generate a VALID result. The formal proof and a synchronization mechanism are given in [12], where we have also analyzed the convergence time. The key idea is the existence of a strictly decreasing best cost chain on downstream routers verifying the property (a).
After some experimentations it appears that it is not useful to try do validate as many next hops as possible. The more DT stores candidate NHs, the more difficult it is for DT(p), with $p > 1$, to verify the property (a). We choose as an improvement to try to validate only all best cost next hops and the best valid sub optimal next hop if any (DT has only to store these NHs). This insures that in most cases there will be at least one alternate path. The set of routes generated by this improvement is not a subset of routes validated without modifying DT and vice-versa.

B. Example

In figure 2, the path $P_1(S,D)$=*S-1-D* is the best path linking S and D, whereas $P_2(S,D)$=*S-2-4-D* is a simple transverse path, $P_{ft}(S,D)$=*S-2-3-1-D* is a forward transverse path and there is no backward transverse path linking S and D. Edges $\{2,3\}$, $\{4,D\}$ and $\{5,D\}$ are transverse in the best path tree rooted on S. *S-2-3-D* is not a transverse path because the edge $\{3,D\}$ connects two paths of the same branch: $branch_1(S)$. This link is ignored in the DT computation by *S*. However, node 2 may use this edge, thanks to hop by hop routing, for its upstream node S in order to form a route. DT computes all paths containing at most one transverse edge. In the same example, DT finally only stores the best path *S-1-D* for the outgoing interface towards *1* and a transverse path *S-2-4-D* for the outgoing interface *2*.
With DT(1), if we consider an uniform link valuation, six routes are validated from S to D : *S-1-D*, *S-1-3-D*, *S-2-3-D*, *S-2-4-D*, *S-2-4-5-D*, *S-2-3-1-D*. To illustrate the NH composition, let us consider the path computed on 2 with DT. Router 2 computes a path via $S = NH_3(2,D)$ to reach D, so that packets may loop on the link $\{S,D\}$. Indeed, router 2 has three candidate routing rows corresponding to its best path *2-3-D*, a transverse path *2-4-D* and a backward transverse path *2-S-1-D*. However, 2 does not validate the last path for S as an incoming interface, but it validates this next hop only for its local traffic, $NH(2,2,D)$, to form the route $R_3(2,D) = \{S,1,D\}$.
Thanks to DT(3), *2* can use the alternate route *2-3-D-4* to reach *4*. On the router *3*, the path via *D* does not satisfy condition (1). Then, DT(3) explores the router *D* which has two solutions

to reach *4* (because of DT modification to improve DT(p)). $NH_1(D,4) = 4$ satisfies condition (1) but $NH_2(D,4) = 5$ does not. Thus, *D* sends $query(2,4,1,1,\{3,D\})$ to *5*. $NH_1(5,4) = 4$ produces a VALID result and $NH_2(5,4) = D$ generates a SKIP result. *5* send a VALID reply to *D* which do the same with *3*. *3* validates a routing row $(2,4,D)$ for transit traffic coming from *s* because $(3,4,D)$ is activated ($C_2(3,4) = C_1(3,4)$) and informs *2* which validates the routing row $(2,3,D)$ for its local traffic.

C. Load balancing

This section presents a formulation of the load balancing issue and a proposition in a distributed multipath context. Load balancing is common in ISP networks. There exists several theoretical propositions, but only the simplest ones are used in real environments. We tried to find a compromise between computational overhead and reactivity. Indeed, according to the time scale of our measurements to analyze the network activities, it is very difficult to quickly react to fast and strong load oscillations using a complex algorithm.
In this paper we consider load balancing among variable rate flows such as TCP flows.

1) TE module: Bandwidth measurement may allow to prevent congestions. We choose a load balancing scheme which favors the minimal cost path utilization until significant trouble occurs. Our TE load balancer prevents routers from recalculating unnecessary proportions so often that it could lead to unwanted oscillations. In our context, the objective of real time measurements is to produce a set of proportions corresponding to the quantity of load to share among several next hops for the same destination. When the network is weakly loaded, it is preferable that routers only use one of their best next hop (the minimal cost path) in order to use less resources. This set of proportions is associated to a specific destination and has to verify several conditions.
Formally, we denote, $\{x_1^d, x_2^d, ..., x_j^d, ..., x_n^d\}$ the vector of global (local and transit traffic) proportions according to a destination *d* on a router *s* which has *n* possible next hops to reach *d*. *j* denotes the rank of the path according to metric C given in table I. We also denote $\{x_1^d(p), x_2^d(p), ..., x_j^d(p), ..., x_n^d(p)\}$ the proportions of traffic coming from interface *p* sent via NH_j. Note that if NH_j is not a valid next hop for traffic coming from *p*, its proportion is set to *0*. We denote $V_d(p)$ and V_d^T respectively, the load coming from *p* and the total load aimed at destination *d*. *I* denotes the set of potential upstream routers to s related to destination *d*.
These variables are subject to the following constraints:

$$\sum_{j=1}^{n} x_j^d = 1 \tag{2}$$

$$\forall p \in I \quad \sum_{j=1}^{n} x_j^d(p) = 1 \tag{3}$$

$$\forall j \in 1, ...n \quad \sum_{p=1}^{k^-(s)} \frac{x_j^d(p) \times V_d(p)}{V_d^T} = x_j^d \tag{4}$$

Equations (2) and (3) imply the consistency of global and per incoming interface proportions. Equation (4) indicates that the sum of incoming proportions reported to the quantity of traffic they have to support depends on global proportions.
We define the function $U(l)$, where *l* an outgoing link of *s*, as the total traffic on link *l* divided by its capacity c_l :

$$U(l) = \sum_{p \in I, d \in N}^{\{j | NH_j(s,d) = l.x\}} \frac{x_j^d(p) \times V_d(p)}{c_l}$$

where $x_j^d(p)$ corresponds to the portion of traffic coming from p which is sent via the link l towards d.
A global heuristic to improve network usage is to minimize the maximum link utilization in the network, i.e:

$$\min \max_{l \in L} U(l) \tag{5}$$

The objective of this optimization problem is to anticipate congestion by minimizing the load of highest loaded links. The idea is that when links are weakly loaded, network response time is globally better. Such a formulation implies linear programming in order to optimize this global objective function. This is unsuitable for a distributed and high performance computation, especially to produce quick local decisions when the traffic is unpredictable. We choose to use a purely local incremental heuristic to approach desired proportions. First, we decide to not consider destination and incoming interface in our measurements, in order to reduce complexity. Each router s only needs to measure the load, denoted $u_l = U(l) \times c_l$ during a chosen time scale *t*, for each of its outgoing links.
The load balancer reacts only when a link *l* is stressed according to a given threshold: $u_l > \alpha \times c_l$. Actually, the choice of a time scale is a fundamental issue. For example, a gigabit incoming traffic can fill up a queue of *600 000* bits (*75* packets of *1000* bytes) in *0.6* milliseconds. The monitoring period should be strictly smaller than the millisecond to compute new proportions which allow to dynamically avoid loss. So we aim at preventing only persistent congestions during more than a second. With our load balancing scheme, each router only needs to compute the load of each of its links and to determine which one carries the most critical load. Our TE-module shifts the amount of demands on an alternate next hop if its own traffic is low enough. After each monitoring period *t*, a router s chooses (if there are multiple local congestions) its worst link in terms of load and moves a part of this load to the best non-stressed NH (a NH is considered as non-stressed if $u_l < \beta \times c_l$). A link *l* corresponds to a j^{th} NH ($l.x = NH_j(s,d)$) for a set of couples destination/incoming interface (d,p). We use the following formula to compute the relaxed proportion on the most stressed link l after each monitoring period *t* :

$$\forall p \in I, d \in N \quad x_j^d(p) \leftarrow x_j^d(p) \times (\frac{\alpha \times c_l}{u_l}) \tag{6}$$

If a congestion occurs on a link *l*, our load balancer shifts, for every possible destination and incoming interface (if there

exists a local non stressed alternative), the corresponding proportion $x_j^d(p)$ to an alternate NH. In practice, to adjust proportions, we take into account the NH capacity and the associated alternate route cost. The intuitive idea is to incrementally reach proportions approximating the min-max problem given in (5) but only when links are really stressed. We also define a mechanism to return to the initial condition when a link l, corresponding to a primary next hop in some routing row, is not stressed anymore. The proportion $x_1^d(p)$ corresponding to a routing row $(p, NH_1(l.x, d), d)$ progressively returns to 1.
It is important to understand that this paper aims to underline the advantage of path diversity generated with DT(p) rather than show performances of our local load balancing scheme. Indeed, our TE load balancer is a simple heuristic and our aim is to put forth the interest of using a good set of paths, in terms of quality and quantity.

2) TCP incidence: Dynamic load balancing needs schemes that split traffic across several paths linking the same pair of routers using a fine granularity. Traffic can be split at two levels. Packet level splitting (for example with a simple round robin scheduling) is well suited to quickly assign the computed load proportion on each path. When paths have different delays, this fine granularity can mis-order a large number of packets. A TCP flow interprets this mis-ordering as a sign of congestion and reduces its congestion window size, resulting in lower performance.
Flow level splitting (a flow can be defined at different levels of granularity: source, destination, port, ...) maps each flow to a specific path and solves the mis-ordering problem. With this type of splitting (often based on hashing schemes), load assignment is unable to accurately and quickly re-balance the load if strong variations in traffic demand occur. FLARE [10] is a compromise between these two solutions. This technique exploits a simple observation on the maximal delay difference between parallel paths and the time between two consecutive packet bursts. It switches packet bursts instead of flows. In this case, load splitting is more accurate and avoids the TCP mis-ordering problem.
In our simulations, we split traffic at the flow level. All packets belonging to a given flow are tagged at the source with a unique random number $n \in [0, N]$. The tag is then similarly exploited by each multipath router to determine the next hop to use. Routers just pick the tag and forward the packet to the path for which the proportion boundaries contain the tag value. This tag could also be computed (by a hashing method for example) and be used differently on each hop. Technically, the IP packet header must contain a tag value belonging to a $\lceil log_2 N \rceil$ bit field. With IPv6 the flow label can easily contain this tag whereas the ToS/DSCP field can carry it in IPv4 . Note that the tag could be used to implement some form of QoS routing. For example, we could decide to force small sized flows to systematically follow the best route. In [17], the authors deliver a detailed analysis of long lived flows routing issues. They propose to route long lived flows on non-minimal routes, but to forward short flows on the shortest path.

IV. Evaluation and simulation

A. Simulation setup

We use Network Simulator 2 [2] (ns2) in order to compare different routing approaches. SPF and ECMP routing protocols are already implemented in ns2 with a link state metric. We have extended ns2 to implement DT(p) and LFI routing and added our TE module to the classifier.

1) Simulation topologies: We present here results obtained on three different topologies. The first two networks (Open transit and Alternet) are topologies measured from traces. We choose these two topologies among a larger set of maps (given in [1]). Alternet represents the category of very meshed network whereas Open transit is less connected. These network topologies have been obtained through the *mrinfo* tool. For networks where native multicast routing is enabled and *mrinfo* is not filtered, this tool gives precise maps of router interconnections (see [16]). For simplicity, we assume symmetry in connectivity and weight assignment and we consider each link to be cost equal. We favor the use of our own topologies because the authors of [18] have found that the Rocketfuel topologies have significantly higher apparent path diversity than what they measure in reality. These topologies both contain false links and miss actual links, and therefore can create an important bias.
We have also simulated the real propagation delays of each link using an orthodromic method allowing us to determine the physical length of each link fairly precisely. For the GEANT topology, we use an additive metric and a link valuation given in [19]. Table IV summarizes the main characteristics of our evaluation networks.

Network name	# of nodes	# of edges	Diameter
Alternet	83	334	8
Open Transit	76	206	11
GEANT	23	74	6

TABLE IV
Evaluation networks

2) Traffic: While most Internet flows are short lived and small sized, the majority of the packets and bytes belong to long lived and big sized flows (see [11] for a detailed traffic analysis). Indeed, flow size distribution as well as flow duration distribution follows a heavy tailed distribution. We have run simulation scenarios under realistic traffic conditions. Our flow generator uses the data sets given in [19]. The sets of traces describe traffic activities by periods of 15 minutes. Each data set consists of traffic matrices built using full IGP routing information, sampled Netflow data and BGP routing information. A traffic matrix gives us the amount of traffic exchanged between each pair of nodes. We have replayed network activities using the information given in the traffic matrix with the finest granularity in order to simulate a real network behavior. It produces coherent traffic quantity according to the given traffic matrix, and it generates a heavy tailed flow size distribution. We use a random number generator to draw a value *m* belonging to $[0,1]$ and we apply the following function $m \rightarrow \frac{x_{min}}{m^k}$ to determine a flow size. The resulting flow size frequencies follow a power law distribution. x_{min} is the minimum flow size and k is the shape parameter. We choose a shape parameter $k = 2$ for our flow size generation to perform a good compromise between intensive flow generation and the running time of our simulations. We do not consider the application level but only the transport protocol level characteristics. Each entry (i,j) in the traffic matrix is decomposed in TCP flows (the minimum flow size is $x_{min} = 20kB$ whereas the maximum flow size is bounded by $10GB$). These flows generate TCP elastic traffic from router i towards router j with a randomly chosen departure time. Each simulation script generates at least *100 000* flows. 80% of the global load is carried by the 20% biggest flows (whose size is superior to $500kB$). We have considered that the impact of very small flows is negligible. In practice, we use drop-tail queues which are able to contain 75 packets of $1kB$ and the sender's TCP window is bounded by 65 packets. The traffic quantity given in [19] data sets is not sufficient to stress (loss and links utilization are very low) the over-provisionned GEANT network, so we increase some entries in the matrix to artificially trigger significant congestions. Results have been obtained on a subset of possible congestion scenarios. We choose to increase the load on links which are already loaded (the load recorded at the time scale of the second exceeds 10% on links with a capacity superior to $2Gb.s^{-1}$). We define and use three models of congestions: $1 \rightarrow n$, $n \rightarrow 1$ and $1 \rightarrow 1$. The first two models increase by a factor of 5 a row or a column of the traffic matrix whereas the third model increases a single entry. In all cases, we choose to generate a single persistent link congestion with SPF routing.

B. Results

1) Path diversity results: First, we have evaluated path diversity depending on the path validation protocol. We have computed the total number of possible routes enabled with DT(3), LFI and SPF. Figure 3 shows the path diversity in terms of loopfree routes. These routes are gathered according to the shortest route which links a pair of routers. This figure compares the ability of each of these techniques to find a large number of alternate routes. Each bar represents (with a logarithmic scale) the number of alternate and shortest routes, grouped by the length of the shortest one. The number of alternate routes is very low with LFI (or ECMP). Actually, when the link valuation is uniform, LFI validates only equal best cost paths. In real topologies, this kind of characteristic is rare. We have also analyzed the capabilities in terms of rerouting. We have measured the percentage of routers able to use an alternative route to circumvent congestions on a primary route. Table V gives a synthetic view of these results. Local rerouting means there exists a local alternative on the router detecting the congestion whereas upstream rerouting means that an upstream router, including up to the source, may redirect the traffic if it receives a warning. Such a difference between LFI and DT(1) for Alternet and Open transit networks is partially due by the uniform link valuation (for a detailed analysis of topology characteristics, refer to [12]). Indeed, on a valuated topology such as GEANT, we observe different results. A depth $p > 1$ allows to benefit from cycles of long size whereas LFI can produce better results than DT(1) if the topology does not permit to take advantage of condition (1).

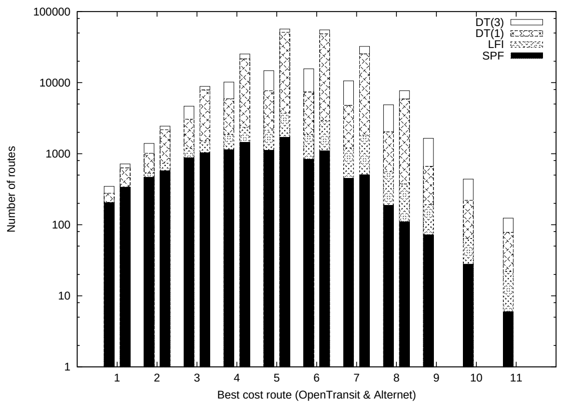

Fig. 3. Routes number

Network name	Local			Upstream		
	LFI	DT(1)	DT(3)	LFI	DT(1)	DT(3)
Alternet	17.8	97.8	99.2	34.2	98.4	99.8
Open Transit	15.9	59.7	77.8	33.3	75.7	91.7
GEANT	36.6	36.6	74.6	63.4	62.4	94

TABLE V
REROUTING CAPACITIES

2) TE results: In this section, we present a set of simulations to evaluate the performance of different path validation techniques associated with our load balancer. We have retrieved two main indicators to compare the efficiency of LFI (with our TE load balancer mechanism), DT(1)-TE and DT(3)-TE compared to SPF routing:

i- Global number of dropped packets.
ii- Link load in the time scale *t*.

Table VI contains our main simulation results. The first line gives the average packet loss reduction ratio compared to SPF. The second line shows the average load percentage of the most loaded link. We discard the warm up period and just consider the steady state period. Average load results on each run have been obtained with a 95% confidence level. The confidence interval is below 0.1% of the link capacity in the steady state period of each run. The parameters used for our load balancer are $t = 1s$, $\alpha = \frac{1}{2}$ and $\beta = \frac{\alpha}{2}$.

Indicators	LFI	DT(1)	DT(3)
average loss reduction ratio	3.8	4.2	6.5
average load of most loaded links (SPF:76%)	61.4	61.4	51.8

TABLE VI
TE RESULTS

We have also measured each TCP flow duration. The difference in duration with SPF is not really significant although in favor of multipath routing under these simulation conditions. DT(1) brings the best results but the difference with LFI is insignificant. We observe a slight deterioration in the duration of diverted flows with DT(3) compared to LFI and DT(1) when using TCP windows of 65 packets. This is due to the use of alternate routes presenting a larger RTT. A larger RTT reduces performance when the TCP window size limits throughput. We notice that, using a larger TCP windows, DT(p), with $p > 1$, can reduce flows duration. DT(p) computes non-optimal routes which can be useful in case of significant trouble (critical congestions or link/router fault). Moreover, results of the table VI represent the mean values of all simulations results. In approximatively 1 simulation out of 5, our TE load balancer simply moves the congestion to another link. In some of these cases, the load balancer does not improve routing. In future works, it will be necessary to define how routers may coordinate local actions in order to provide a more efficient reaction in these particular cases. Figure 4 illustrates the most loaded link utilization with different techniques.

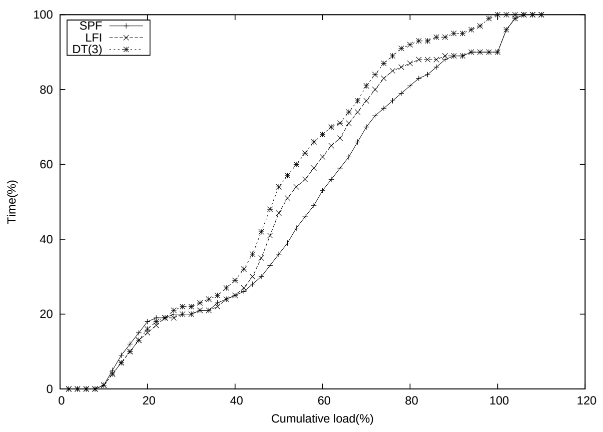

Fig. 4. Cumulative load

The x-axis gives the percentage of load depending on the link capacity. Here, load means the quantity of traffic that the router tries to forward on the considered link at the time scale of t. We use load data collected in each simulation. The y-axis shows the cumulative utilization time during the steady state period. We observe that DT(3) allows to reduce critical link utilization compared to SPF and LFI (DT(1) results are almost equal to LFI results). For example the load is less than 50% during 40% of time with SPF and during 58% of time with DT(3). We also notice that multipath routing permits to use non-stressed links to balance the load. The depth parameter p is a key factor to minimize the maximum link utilization. However, with a large p, the load balancer has to guarantee a careful use of longer alternate routes.

V. Conclusion

Congestion control algorithms can coexist at transport and routing level to balance the load in IP networks. In this paper we proposed a new scheme to improve network response in a distributed way. DT(p)-TE offers the possibility to use temporarily alternative routes in order to reduce packet loss and throughput decrease. Simulation results emphasize the interest of our proposition. The path diversity and the capacity to coexist with an elastic and non stationary fluid model of traffic, such as TCP flow aggregation, seems to be fundamental to perform dynamic routing. We still have to work on the adjustment of the time scale parameter of the load balancer. This issue is essential to trigger an appropriate reaction and implement an efficient local load balancing. We also have to work on the way to coordinate reactions between routers. So they could exchange local informations in order to avoid inappropriate load shifting when local decisions are inefficient. With such a signaling protocol, a router should be able to consider the route quality up to the destination. We are currently investigating the convergence time of a protocol which distributes the load balancing decisions. This problem concerns both congestion avoidance and fast rerouting issues when there is no local solution.

References

[1] “Multicast maps, http://clarinet.u-strasbg.fr/∼merindol/maps.tar.gz.”
[2] “The network simulator- ns2, http://www.isi.edu/nsnam/ns.”
[3] A. Atlas, “U-turn alternates for ip/ldp fast-reroute draft-atlas-ip-local-protect-uturn-03,” IETF, Draft, Feb. 2006.
[4] A. Atlas and A. Zinin, “Basic specification for ip fast-reroute: Loop-free alternates draft-ietf-rtgwg-ipfrr-spec-base-06,” IETF, Draft, mar 2007.
[5] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “Rsvp-te: Extensions to rsvp for lsp tunnels,” IETF, RFC 3209, 2001.
[6] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, “MATE: MPLS adaptive traffic engineering,” in *INFOCOM*, 2001, pp. 1300–1309.
[7] D. Eppstein, “Finding the k shortest paths,” in *IEEE Symposium on Foundations of Computer Science*, 1994, pp. 154–165.
[8] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, and N. Feldman, “Constraint-based lsp setup using ldp,” IETF, RFC 3213, Jan. 2002.
[9] S. Kandula, D. Katabi, B. Davie, and A. Charny, “Walking the tightrope: Responsive yet stable traffic engineering,” in *SIGCOMM ’05*, 2005, pp. 253–264.
[10] S. Kandula, D. Katabi, S. Sinha, and A. Berger, “Dynamic load balancing without packet reordering,” in *ACM SIGCOMM vol.37*, April 2007, pp. 51–62.
[11] K. Lan and J. Heidemann, “On the correlation of internet flow characteristics,” USC/ISI Technical Report ISI-TR-574, Tech. Rep., 2003.
[12] P. Merindol, J. J. Pansiot, and S. Cateloin, “Path computation for incoming interface multipath routing,” in *ECUMN’07*, 2007, pp. 75–85.
[13] J. Moy, “Ospf version 2,” IETF, RFC 2178, Apr. 1998.
[14] S. Nelakuditi and Z.-L. Zhang, “On selection of paths for multipath routing,” in *Proceedings of IWQoS*, 2001.
[15] D. Oran, “Is-is intra-domain routing protocol,” IETF, RFC 1142, 2001.
[16] J. J. Pansiot, “Local and dynamic analysis of internet multicast router topology,” *Annals of telecommunications*, vol. 62, no. 3-4, 2007.
[17] A. Shaikh, J. Rexford, and K. G. Shin, “Load-sensitive routing of long-lived ip flows,” *SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 215–226, 1999.
[18] R. Teixera, K. Marzullo, S. Savage, and G. M. Voelker, “In search of path diversity in isp network,” in *ACM IMC ’03*, October 2003.
[19] S. Uhlig, B. Quoitin, S. Balon, and J. Lepropre, “Providing public intradomain traffic matrices to the research community,” *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, January 2006.
[20] S. Vutukury, “Multipath routing mechanisms for traffic engineering and quality of service in the internet,” Ph.D. dissertation, University of California, Santa Cruz, Mar. 2001.
[21] X. Yang and D. Wetherall, “Source selectable path diversity via routing deflections,” in *SIGCOMM vol.36*, october 2006, pp. 159–170.