Adaptive Probabilistic Flooding for Multi-path Routing

Christophe Betoule, Thomas Bonald, Remi Clavier, D. Rossi, Giuseppe Rossini, Gilles Thouenon

Adaptive Probabilistic Flooding for Multipath Routing

Christophe Betoule[2], Thomas Bonald[1], Remi Clavier[2], Dario Rossi[1], Giuseppe Rossini[1,†], Gilles Thouenon[2]

[1] Telecom ParisTech, Paris, France
firstname.lastname@telecom-paristech.fr († corresponding author)
[2] Orange Labs, Lannion, France
firstname.lastname@orange-ftgroup.com

***Abstract*—In this work, we develop a distributed routing algorithm for topology discovery, suitable for ISP transport networks, that is however inspired by opportunistic algorithms used in ad hoc wireless networks. We propose a plug-and-play control plane, able to find multiple paths toward the same destination, and introduce a novel algorithm, called *adaptive probabilistic flooding*, to achieve this goal. By keeping a small amount of state in routers taking part in the discovery process, our technique significantly limits the amount of control messages exchanged with flooding – and, at the same time, it only minimally affects the quality of the discovered multiple path with respect to the optimal solution. Simple analytical bounds, confirmed by results gathered with extensive simulation on several topologies (up to 10,000 nodes), show our approach to be of high practical interest.**

I. Introduction

Scaling layer-2 protocols to very large networks [1] let enterprises deal with simpler and less expensive hardware, which translates in lower capital expenditures and management costs. Every solution proposed so far tries to get, from one side, a scalable architecture in which packets are routed on the shortest path; on the other side, such architecture should be flat [2], self-configuring and possibly self-healing [3]. Many works address this tradeoff, proposing different routing and forwarding schemes, trying to seamlessly fill the gap between local and wide area protocols: Viking [4], SEATTLE [5], PortLand [3] represent few recent examples of research efforts in this direction.

The routing process of such architectures is usually divided in two parts: the *host routing* process [5] aims at resolving each host to a single switch; then, the *switch routing* process let each core machine learn the best path to the other switches [6]. The latter process, to which we focus in the following, is commonly addressed [3]–[5] with a link-state algorithm (similar to OSPF or IS-IS) implemented at layer-2, and that generally yields a single, shortest, path to any other switch (with the exception of [4] that also keeps a backup tree).

We believe that using a link state algorithm at layer-2 is not the best choice. First, link state decouples topology distribution and routes computation –in the sense that all topological information needs to be received by all nodes prior that the routing table can be computed– which represents a useless waste of time. Then, notice that the core routing table computation algorithm is represented by Dijkstra algorithm, that results in a $O(NlogN)$ complexity. However, Dijkstra only provides a single shortest path between any nodes pair, while additional paths computation produces even higher computational complexity (though still polynomial).

In this work, we propose Adaptive Probabilistic Flooding (APF) as a simpler solution to the aforementioned problem of switch routing, which takes an alternative approach for the computation of multiple disjoint paths. Aiming at simplicity, we devise a distributed greedy algorithm that drops computational complexity still remaining (almost) stateless and self-terminating. In our design, simplicity tradeoffs with the communication cost, as APF generates an higher number of messages with respect to link state algorithms. At the same time, we can tune (and analytically bound) the number of messages thanks to a simple parameter that drives the speed of the probabilistic procedure. Moreover, by means of simulation on several topologies (up to 10,000 nodes), we show that, with as few as 2-3 times more messages that a link state algorithm, APF is able to find the shortest path and optimal backup path in more than 90% of the cases. In the remaining 10% of the cases, non-optimality is due to a limited amount of overlap between primary and secondary paths (about 1 node on average).

The rest of the paper is organized as follows. Related work is presented in Sec. II. The Adaptive Probabilistic Flooding (APF), a path discovery algorithm, is described in Sec. III. Sec. IV is devoted to the performance analysis, where we provide analytical bounds for the number of messages, and simulation results for the APF path quality. Finally, Sec. V concludes the paper and outlines future work.

II. Related work

Our work focuses on two seemingly conflicting aspects: (i) abating routing algorithms complexity, to make them suitable for layer-2 devices (e.g., Ethernet-like switches), and (ii) enabling multi-path features. In this section, we explore the closest related work in both areas.

The first topic, i.e., the desing of light-weight routing protocol has been addressed mostly in wireless area [7]–[10]. DSR [7] greedy determines routes by means of packet flooding, employing a TTL to limit the flooding depth. When

a control message reaches the destination, it is sent back to the origin that thus discovers a path toward the target. In sensor-networks, SCOUT [8], routing in a hierarchy of sensors, follows a classical landmark routing [9] scheme. In VRR [10] instead, the whole topology is arranged as a DHT, and the forwarding is done accordingly. However, solutions like [7], [8], [10] rely heavily on the notion of broadcast, so they are not directly applicable in a wired environments. Among them, the closest approach to ours is [7], which does not offer multiple paths and is moreover critically affected by TTL tuning (unlike our proposed APF approach, as we shall see in Sec. III).

The second branch of work close to ours [11]–[18] addresses the multipath problem. In wireless networks, a multipath extension to DSR is proposed in [11], where authors employ a reactive flooding algorithm, in which the TTL is gradually increased in order to find more than just one single response. However, while gradual TTL increase is able to provide multiple paths and is more robust to wrong TTL settings, it brings two shortcomings, namely a (i) slower convergence of the routing process, and a (ii) higher number of control messages. In wired network, the multipath problem is often treated jointly with traffic routing in the data plane, solving a multi-commodity flow problem [12] where, given a traffic matrix and a topology, the objective is find the routing that minimizes network congestion. Another class of work closest to ours focuses instead on control-plane topology discovery [13]–[18]. The simplest link state extension to multipath is Equal Cost Multiple Paths (ECMP) [13], that splits traffic on different shortest paths (if there exist) between each pair of nodes. Other works relax the hypothesis of equal cost, and develop algorithms to find k shortest paths on any given graph [14], [15]. Addressing multiple different paths (i.e., not necessarily the k shortest ones), [16] proposes a link state algorithm, modifying the downstream criterion to avoid loops. Notice however that, while path computation still faces the delay due to link state advertisement, the computation of alternative paths also adds further complexity beyond the $O(NlogN)$ Dijkstra cost (respectively, $O(E+NlogN+k)$ in [14], $O(k(E+NlogN))$ in [15] and $O(E^2)$ in [16]). Finally, a different approach from link state algorithms is taken, e.g., in [17], [18], that employs distributed Ant Colony Optimization (ACO) algorithms, based on swarm intelligence. In [17], [18] a set of ants is spread through the network in order to discover disjoint multiple paths, and the pheromone left by the ants is employed in order to avoid already crossed paths. In this case however, the shortcoming is that the routing process is not guaranteed to converge to the shortest path (unlike APF).

III. Path discovery algorithm

In this work, we propose Adaptive Probabilistic Flooding (APF), a simple algorithm for multi-path topology discovery. Summarizing the main differences with related literature, APF sits at a different operational point in the tradeoff between algorithmic complexity vs. control message overhead. Unlike in link state algorithms [13]–[16], since in our approach the whole traveled path accumulates in the control message as in [7], [11], every message brings useful information for path discovery, that can thus be computed online. At the same time, unlike in [7], [11], our algorithm does not rely on critical parameters such as TTL. Furthermore, the actions that need to be taken to handle each message are simple operations, making thus the algorithm viable on simple hardware. Finally, unlike [17], [18], the algorithm is guaranteed to find the shortest path, and as our simulation results confirm, the secondary path is very often the optimal one found by [14], [15] link state algorithms.

A. Overview

We aim at designing a distributed algorithm for path discovery, capable of finding multiple, possibly disjoint paths between any pairs of nodes. To do so, each node periodically advertises its presence by means of some flooding procedure described below. Specifically, each node sends an *advertisement* message every τ_a seconds; typical values range from a few seconds to minutes [19].

During the flooding procedure, each relay node adds its identifier to the advertisement messages it receives, so that these messages carry information concerning the whole traveled path. Upon the reception of an advertisement message, a node learns a path from the source of this message, as well as from any intermediate node on this path, as in [7]. Flooding decisions are taken independently by each node, and constitute the core of the algorithm. The main idea is that nodes need to flood a received message *at least once*, so that shortest paths are discovered. Nodes actually need to flood the message *multiple times*, in order to discover further paths beyond the shortest one. The number of flooding decisions is critical with respect to both the quality of the path discovery and the overhead of the algorithm.

A simple option [7], [11] could consist in including a Time To Leave (TTL) field in the packet, so as to interrupt the flooding process when some pre-configured maximum path length is reached. The selection of a proper TTL value is critical in this case: if the TTL is shorter than the graph diameter D for instance, then connectivity cannot be guaranteed; if the TTL is too large, the overhead of the algorithm becomes prohibitive (as the number of relayed messages is exponential in the TTL).

We propose an alternative approach based on *adaptive probabilistic flooding*. Any node receiving some advertisement message from source s floods this message the first time, and floods it with some decreasing probability the following times. Specifically, node i floods an advertisement message generated by source node s over all its links (except the one from which it has received the message) with probability:

$$P = \beta^{n_{i,s}} \tag{1}$$

where β is some fixed parameter and $n_{i,s}$ is a counter, stored at node i, of the number of times node i has already received an advertisement originated by node s. The flooding decisions are taken independently on each link, and the counter is reset periodically, as explained later. Note that node i floods the

first advertisement message it receives for source node s since $n_{i,s} = 0$ in this case. As further messages are received, flooding will become exponentially less likely, according to the backoff parameter β. Note also that, if we set $\beta = 0$ APF produces the same number of messages of a link state algorithm for propagating the network topology. The quality of the path discovery is expected to increase with β, at the expense of larger overhead. However, we shall see that performance is not very sensitive to this parameter, and rather degrades gracefully with parameter misguidance, which makes the algorithm robust and practically interesting.

B. Primary and secondary paths

Consider a network, modeled as an undirected graph $G = (E, V)$, composed of $|V| = N$ routers, in which any pair of adjacent routers are connected by a single link for simplicity (the algorithm can be easily extended to the general case of multiple links between any pair of nodes). Between any two routers $i, j \in V$, we are interested in finding a pair of *paths*, i.e., sequences of edges connecting node i to j. We denote by $\mathcal{P}$ and $\mathcal{S}$ the primary and secondary paths, respectively, returned by the adaptive probabilistic algorithm on graph G [1]. We denote by L_p and L_s the respective lengths of these paths.

To gauge the quality of the primary and secondary paths found by our algorithm, we need to define target path properties. The primary path is expected to be the shortest path in number of hops; in other words, we say that $\mathcal{P}$ is optimal if it belongs to the set of shortest paths from i to j in G (as there may be several such paths). The secondary path is expected to minimize the similarity with the primary path, $\mathcal{P} \cap \mathcal{S}$. Note that this choice reduces the share of faith between these paths, improving network resilience against failures and traffic surges.

To find the optimal secondary path $\mathcal{S}$, we consider a modified graph G' in which the cost of links along the primary path $\mathcal{P}$ are increased by the network diameter [20], and other link costs are unitary. As links belonging to $\mathcal{P}$ are now discarded due to higher cost, running Dijkstra on G' we retrieve a path $\mathcal{S}'$ minimizing the similarity function $\mathcal{P}' \cap \mathcal{S}'$ (notice that since nodes along the primary path are not removed from G', they can be included in $\mathcal{S}'$ only if strictly necessary as the path would otherwise be disconnected). We say that the secondary path found by the algorithm $\mathcal{S}$ is optimal if $|\mathcal{P} \cap \mathcal{S}| = |\mathcal{P}' \cap \mathcal{S}'|$ and $L_s = L_{s'}$, i.e., the length L_s of the secondary path is equal to the length $L_{s'}$ of the optimal $\mathcal{S}'$ (as there may be multiple disjoint paths minimizing the similarity with the shortest path).

C. Pseudocode

A pseudocode description of the algorithm is given in Fig. 1. A source node s initiates the advertisement process by flooding an advertisement packet ADV to all its neighbors. The flooded packet contains a list of node identifiers ID, initially set to ID[0]=s by the source, to which each node appends its own identifier. Upon reception of an advertisement packet ADV,

[1]The aggregation of primary and secondary paths at a given node, forms two distinct trees of the network, as in [4].

```
1: while {receiving message ADV} do
2:   ℓ ← length(ADV.ID)
3:   for all { i ∈ [0, ℓ] } do
4:     if {ADV.ID[i] = j} then
5:       exit // Break loop and abort flooding
6:     else
7:       d ← ADV.ID[i] // Destination
8:       L_{j,d} ← (ADV.ID[ℓ], ..., ADV.ID[i])
         // Overhearing advertised paths from ADV;
9:       if { ∄P_{j,d} ∨ length(L_{j,d}) < length(P_{j,d})}
         then
10:        P_{j,d} ← L_{j,d} // Update primary path
11:      end if
12:      if {∄S_{j,d} ∨ |P_{j,d} ∩ L_{j,d}| < |P_{j,d} ∩ S_{j,d}| ∨
         (|P_{j,d} ∩ L_{j,d}| = |P_{j,d} ∩ S_{j,d}| ∧ length(L_{j,d}) <
         length(S_{j,d}))} then
13:        S_{j,d} ← L_{j,d} // Update secondary path
14:      end if
15:    end if
16:  end for
17:  append j to ADV.ID
18:  s ← ADV.ID[0] // Source
19:  for all {next ∈ neighbors(j) } do
20:    if { next ≠ ADV.ID[ℓ − 1] } then
21:      send ADV to next w.p. β^{n_s}
         // Adaptive probabilistic flooding
22:    end if
23:  end for
24:  n_s++ // Update counter associated with source s
25: end while
```

Fig. 1. Algorithm pseudocode for a generic node j of the network

a node learns a (backward) path to the source s and to any intermediate node $d = \text{ADV}.ID[i]$ along the path. In case the receiver j detects a loop (finding its identifier within the ID list), it discards the message and aborts the flooding procedure. Otherwise, it analyzes, and possibly stores, the newly learned path $\mathcal{O}_{j,d}$. Specifically, the primary (and secondary) path is first set if not existent yet. Also, if the newly overheard path is shorter than the primary path $length(\mathcal{L}_{j,d}) < length(\mathcal{P}_{j,d})$, then the primary path is updated with the overheard one. Similarly, if the overheard path has lower similarity than the current secondary path $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| < |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}|$, or if it has equal similarity but is shorter than the secondary $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| = |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}| \wedge length(\mathcal{L}_{j,d}) < length(\mathcal{S}_{j,d})$, then the secondary path is updated.

Finally, after having added its own identifier to the ADV.ID, the node probabilistically floods the ADV message, with independent decisions per each neighbor (except the node ADV.ID[ℓ-1] from which the message came), and updates the per-source counter n_s. Not shown in the pseudocode for the sake of simplicity, ties in the secondary path selection are broken at random.

Notice that, we expect (i) messages on the *shortest path* to reach a node *before* messages that take longer paths. This definitively holds in case of homogeneous delay; otherwise, it may happen that (ii) messages traveling along the *quickest path* arrive first, which would be then stored as primary path;

TABLE I
TOPOLOGICAL PROPERTIES OF THE NETWORK SCENARIOS

Network	Segment	N	δ	σ	D	D'
Tiger2	metro	22	3.6	0.6	5	6
Geant	aggregation	22	3.4	1.4	6	10
Abilene	core	11	2.6	0.5	5	8
DTelekom	core	67	10.38	13.31	3	4
Random	synthetic	≤ 10000	4	≈ 2	-	-

analogously, (iii) if control messages queue with data-plane messages without priority, the first message could have as well traveled along the *less congested* path. Notice that, by simple priority queuing, case (iii) can be ruled out. Then, notice that (ii) may only happen in networks having links with very long delays: in this case, the message traveling along the shortest path is not necessarily the first to be received. However, one of the subsequent messages will surely have traveled along the shortest path (since due to $n_s = 0$ the messages are flooded at least once), so that the primary path is guaranteed to converge to the shortest (unlike [17], [18]).

IV. PERFORMANCE EVALUATION

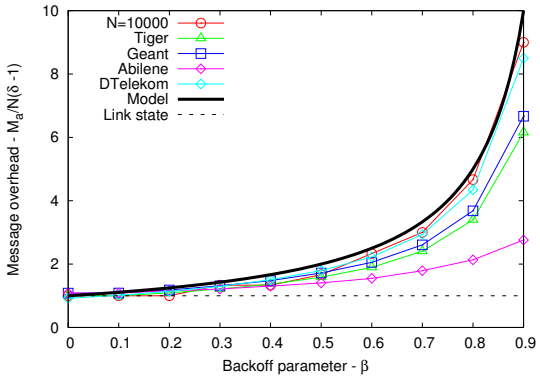
In this section, we precisely quantify the APF *overhead* vs *path quality* tradeoff, and that can be tuned through the β back-off parameter earlier introduced, by proceeding as follows. We first evaluate the overhead of the algorithm through a simple analytical model, then, we employ discrete event simulation to validate the analysis and evaluate performance in terms of the quality of discovered paths.

Simulations are carried on with Omnet++ [21] over four different network topologies gathered by mean of Rocketfuel [22], whose most significant properties are summarized in Tab. I. Specifically, Tab. I reports the number of nodes N, the average and standard deviation of the node degree, δ and σ, the diameter D of the original graph G and the largest diameter D' over the modified G' graphs.

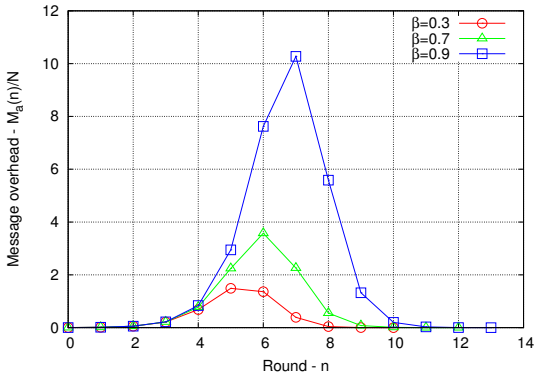
Note that we consider both real ISP topologies, corresponding to different network segments, as well as a set of 50 synthetic random graphs with $N \leq 10000$ and $\delta = 4$. For the time being, we use homogeneous settings (i.e., constant and equal delay on every link), and no failures happen within the network. The evaluation of more complex (heterogeneous delay, failures, etc.) scenarios is part of our ongoing work.

A. Overhead

We now evaluate the cost of APF in terms of communication overhead, space and computational complexity. The average amount of messages handled by any given node during the advertisement procedure is denoted with M_a and represents the primary cost sustained by the algorithm. For comparison purposes, we take any link state algorithms like IS-IS or OSPF as a reference. We recall that such protocols are composed by two phases: (i) flooding of topological information, and (ii) path computation, usually fulfilled by the mean of Dijkstra algorithm on the graph information gathered on the first phase.



(a) Communication overhead w.r.t. link state algorithms, and comparison with analytical bound



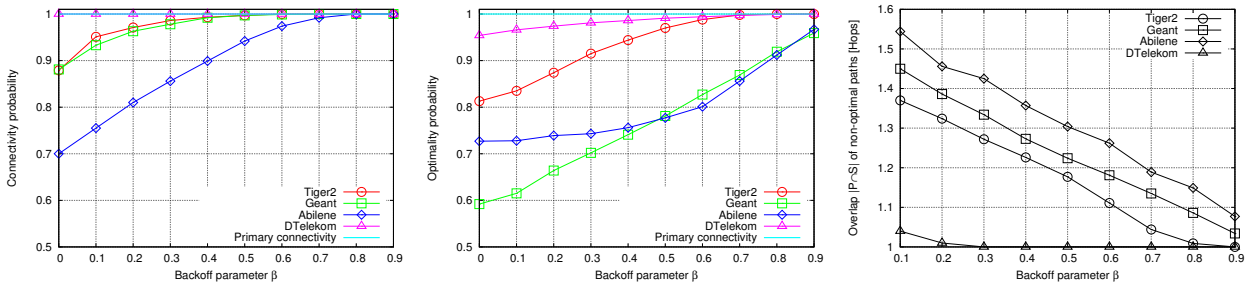
(b) Time evolution of the number of advertisement messages seen by a single node and generated by a single advertisement process ($N = 1000, \delta = 4$)).

Fig. 2. Adaptive Probabilistic Flooding: Communication Overhead

Communication overhead: Consider a single advertisement from some source node s, and consider some relay node j with degree δ. The first time node j receives an ADV message originated by s, it sends a copy on each output link, except the one where the ADV message has been received. This generates $\delta - 1$ messages. The second time j receives an ADV message from the same source, it will forward the message over each of the $\delta - 1$ links with probability $\beta < 1$; so, at second reception, node j generates $(\delta - 1)\beta$ messages on average. Iterating and taking into account N advertisement processes (one per each node), we bound the total number of control messages M_a that are seen by the average node:

$$\begin{aligned} M_a &\approx N[(\delta-1) + (\delta-1)\beta + (\delta-1)\beta^2 + \cdots] \\ &= N(\delta-1)\sum_{n=0}^{\infty}\beta^n \\ &= N(\delta-1)\frac{1}{1-\beta} \end{aligned} \quad (2)$$

It turns out that this simple and conservative bound matches very well the empirical results found by simulation. It's worth to note as the bound in (2), when $\beta = 0$, represents the number of messages a link state algorithm would flood in



(a) Connectivity probability of primary and secondary paths

(b) Optimality probability of primary and secondary paths

(c) Average overlap between primary and secondary path $\|\mathcal{P} \cap \mathcal{S}\|$ for non-optimal secondary paths overlap

Fig. 3. Adaptive Probabilistic Flooding: Path Quality.

order to distribute the whole topology (assuming that $0^0 = 1$ so that messages are flooded exactly once). In this sense we can regard APF as an extension of link state algorithm, β being a parameter which tradeoffs between paths quality and number of additional messages.

Note also that, as the first flood is always performed, convergence of the primary $\mathcal{P}$ path to the shortest path is always guaranteed. Hence, the backoff parameter β affects only the quality of the secondary path $\mathcal{S}$: by tuning β, we can upper bound the algorithm overhead M_a while matching the required level of path quality.

Fig. 2(a) depicts, as a function of β, the upper bound (2) normalized over the number of messages produced by the link state flooding, along with M_a gathered by simulations carried on random and real networks. Notice that, for $\beta \leq 0.7$, APF sends at most 2-3 times more messages than link-state algorithms. Notice also that the bound in (2) is generally a close estimation of the number of messages. Finally, notice also that APF can be applied to fairly large networks, as our simulations up to N =10,000 nodes suggest.

Fig. 2(b) plots the load of a single node for a single advertisement in function of the (slotted) time for a random topology with N =1000 nodes: we observe that, after some initial exponential growth due to the flooding process, the backoff factor kicks in and slows down the growth, which then dies out very fast, due to an increasing number of flooding paths being probabilistically cut out. This auto-termination feature is a very desirable property of the algorithm, and further suggests that advertisement periods do not need to overlap (as in case of *parallel* advertisements), but can rather be *serialized*. In this way, we would tradeoff also between nodes peak load (which increases with the degree of parallelization) and speed of routing convergence (that decreases with the degree of serialization). We remark that serialization is not feasible for link state solutions, as paths are computed just when the whole topology is fully known.

Space complexity: Space needed by APF to run is $O(2N)$ in order to store primary and secondary/backup tree of the network (as in [4]). Additionally, we need also $O(N)$ counters n_s to safely count the number of messages generated by each source. This could be problematic in case the size of the network is not known a priory (though N could be set to a fairly large number for safe operation). At the same time, a staggered advertisement solutions as proposed above, implies that instead of keeping $O(N)$ counters (one for each source in case of advertisements *in parallel*), the system could perform advertisement *in series* and keep a small number of $O(1)$ counters. This could be achieved by desynchronizing the start of each advertisement either with a simple policy (e.g., periodically at random within $[0, 2\tau_a]$) or with more sophisticated schemes (e.g., CSMA-like solutions). This is an interesting direction for future research, that we aim at pursuing in the following.

Computational complexity: If Dijkstra and the other graph algorithms need to perform $O(NlogN)$ and $O(NlogN + E^2)$ for the computation of the shortest and backup path respectively once the full graph is known, APF on the other hand needs to perform simpler operations on a packet-by-packet basis. More precisely, on the reception of each control message, switches need to perform: (i) a comparison for the shortest path (Fig. 1, line 9); (ii) an intersection for the best alternate path (Fig. 1 line 12). Since the overall number of APF control messages is bounded, we can bound APF computational complexity as well – which grows with N when all advertisements start at the same time. Instead, it's worth to note that in link state algorithms, a single link state advertisement for a topology change causes to start, for each node, another run of a $O(NlogN)$ Dijkstra algorithm.

B. Path quality

Let us now focus on the quality of the paths that the adaptive probabilistic flooding algorithm is able to find. For simplicity, we let each node advertise itself once at time $t = 0$ and evaluate the connectivity and optimality of the primary and secondary paths. Since evaluating path quality of random networks is unrealistic, we now only consider the ISP topologies, reporting results over 20 simulations per topology.

We express path quality in terms of *connectivity* along the primary and secondary path (i.e., whether paths $\mathcal{P}$ and $\mathcal{S}$ joining any two nodes $i, j \in V$ exist) and *optimality* (i.e., whether $\mathcal{P}$ and $\mathcal{S}$ are optimal according to the above definitions). We

express connectivity in terms of the probability that, $\forall i, j \in V$, nodes i and j are connected by some primary/secondary path. We express optimality in terms of the probability that the primary path is also the shortest, and that the secondary path is the shortest most diverse path from the primary.

Fig. 3(a) depicts the *connectivity* probability of the primary and secondary paths as a function of β: as expected, primary connectivity does not depend on β and is always guaranteed. Since a primary path is always found, the connectivity index is relevant for the secondary path only: we see that all secondary paths are connected in all networks when $\beta \geq 0.7$ (which correspond to limited overhead in Fig. 2(a)).

Fig. 3(b) reports the *optimality* probability of the primary and secondary paths as a function of β: again, since the shortest path is always eventually found, the optimality of the primary path is guaranteed. Thus, the optimality index is relevant only for the secondary path: we see that a significant percentage (from 60% to 85%, depending on the topology) of secondary paths are optimal even for a very low value of $\beta = 0.1$, and that at least 90% of secondary paths are optimal for all considered topologies when $\beta \geq 0.8$. Moreover, we observe that optimality gracefully degrades β, and furthermore with similar (roughly linear) slope across all topologies. This is a desirable behavior: as no phase transition nor knee appear in the path quality slopes, tuning β between low overhead (low β) vs high path quality (high β) is not critical.

Finally, we dissect the reason behind the sub-optimality of some secondary paths. Recall that a secondary path is optimal if it is the shortest and most diverse path compared to the primary. Hence, sub-optimality of the secondary path may be due to either (i) a non-zero *overlap* between primary and secondary paths, $|\mathcal{P} \cap \mathcal{S}| > 0$, or (ii) a path with a stretch over the optimal secondary path larger than one $L_s/L_{s'} > 1$. Fig. 3(c) depicts the overlap, i.e., the number of nodes that primary and secondary paths have in common, conditioning over the sub-optimal paths (i.e., the overlap of optimal secondary paths is not accounted for in the picture). As shown by the figure, sub-optimality seems to be tied to slightly more than one node in common as $|\mathcal{P} \cap \mathcal{S}| \in [1, 1.5]$. Furthermore, as the average overlap is always $|\mathcal{P} \cap \mathcal{S}| \geq 1$ for any β, we can conclude that overlapping paths are significantly more common that long-stretching paths.

V. Conclusions

We have presented a novel flooding based algorithm for multiple-path discovery: the algorithm trades a small amount of state in routers, i.e., $O(N)$ counters, in order to significantly limit the number of messages generated by flooding through an adaptive probabilistic algorithm.

Simple analytical bounds, confirmed by simulation results, show the overhead entailed by the advertisement procedure to be low (with respect to the amount of messages needed by classical link state algorithms) and auto-terminating (due to the multiplicative decrease of the flooding probability).

Simulation results also testify excellent performance in terms of path quality: connectivity and optimality of the primary path are achieved by design, while 90% of secondary paths are also optimal when $\beta \geq 0.8$ (or otherwise decrease linearly for lower β). Interestingly, the low percentage of low path is due to a very limited amount of share of faith between paths. As part of our future work, we want to carry on more realistic experiments on a wider set of topologies and further reduce the amount of state to $O(1)$, by exploring policies for the serialization of node advertisement.

Acknowledgement

The work presented in this paper has been carried out at LINCS (http://www.lincs.fr) and was funded by an Orange Labs grant.

References

[1] Andy Myers, T.S. Eugene Ng, and Hui Zhang, "Rethinking the service model: Scaling ethernet to a million nodes," in *ACM Hotnet*, 2004.

[2] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *PAM*, Berlin, Heidelberg, 2009.

[3] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, New York, NY, USA, 2009, pp. 39–50.

[4] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh, "Viking: a multi-spanning-tree ethernet architecture for metropolitan area and cluster networks," in *IEEE INFOCOM*, Mar. 2004.

[5] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: a scalable Ethernet architecture for large enterprises," in *ACM SIGCOMM*, Aug. 2008.

[6] A. Singla, P. B. Godfrey, K. Fall, and S. Iannaccone, G. Ratnasamy, "Scalable routing on flat names," in *ACM CoNEXT*, 2010.

[7] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," IETF RFC 4728, Feb. 2007.

[8] S. Kumar, C. Alaettinglu, and D. Estrin, "Scalable object-tracking through unattended techniques (scout)," in *IEEE ICNP*, 2000, pp. 253 –262.

[9] P. F. Tsuchiya, "The landmark hierarchy: a new hierarchy for routing in very large networks," in *ACM SIGCOMM*, New York, NY, USA, 1988, pp. 35–42, ACM.

[10] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual ring routing: network routing inspired by dhts," in *ACM SIGCOMM*. 2006, pp. 351–362, ACM.

[11] M. J. Kim, D. H. Lee, and Y. I. Eom, "Enhanced non-disjoint multi-path source routing protocol for wireless ad-hoc networks," in *ACM ICCSA*, 2007, pp. 1187–1196.

[12] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," in *IEEE INFOCOM*, 2001, vol. 3, pp. 1300 –1309.

[13] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Nov. 2000.

[14] D. Eppstein, "Finding the k shortest paths," *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 0, pp. 154–165, 1994.

[15] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, November 2007.

[16] P. Merindol, J.-J. Pansiot, and S. Cateloin, "Low complexity link state multipath routing," in *IEEE INFOCOM Workshop*, april 2009, pp. 1 –6.

[17] Maria J. Blesa and Christian Blum, "Ant colony optimization for the maximum edge-disjoint paths problem," in *EvoWorkshops*, 2004, pp. 160–169.

[18] N. Lin and Z. Shao, "Improved ant colony algorithm for multipath routing algorithm research," in *IEEE IPTC*, oct. 2010, pp. 651 –655.

[19] Srihari Nelakuditi and Zhi-Li Zhang, "On selection of paths for multipath routing," in *IEEE IWQoS*, vol. 2092, pp. 170–184. 2001.

[20] R.G. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *IEEE Information Theory*, vol. 39, no. 2, pp. 443 –455, Mar. 1993.

[21] Andras Varga, "Omnet++ website," 2010.

[22] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *ACM SIGCOMM*, 2002.