Scalable Routing Easy as PIE: a Practical Isometric Embedding Protocol

Technical Report

Julien Herzen
EPFL, Lausanne, Switzerland
julien.herzen@epfl.ch

Cedric Westphal
Docomo Innovations, Palo Alto, CA
cwestphal@docomoinnovations.com

Patrick Thiran
EPFL, Lausanne, Switzerland
patrick.thiran@epfl.ch

***Abstract*—[1] We present PIE, a scalable routing scheme that achieves 100% packet delivery and low path stretch. It is easy to implement in a distributed fashion and works well when costs are associated to links. Scalability is achieved by using virtual coordinates in a space of concise dimensionality, which enables greedy routing based only on local knowledge. PIE is a general routing scheme, meaning that it works on any graph. We focus however on the Internet, where routing scalability is an urgent concern. We show analytically and by using simulation that the scheme scales extremely well on Internet-like graphs. In addition, its geometric nature allows it to react efficiently to topological changes or failures by finding new paths in the network at no cost, yielding better delivery ratios than standard algorithms. The proposed routing scheme needs an amount of memory polylogarithmic in the size of the network and requires only local communication between the nodes. Although each node constructs its coordinates and routes packets locally, the path stretch remains extremely low, even lower than for centralized or less scalable state-of-the-art algorithms: PIE always finds short paths and often enough finds the shortest paths.**

I. Introduction

In the Internet, the tremendous growth of the number of destinations translates into a corresponding growth of the routing tables. The Internet Architecture Board recently recognized the scalability of routing as being *"the most important problem facing the Internet today"* [2]. The core routers need an excessive amount of resource and power to store, maintain and perform lookups in huge routing tables. The amount of traffic exchanged between the routers is proportional to the size of these tables, and the complexity of managing some state for every destination in the network results in convergence problems and instabilities. The arrival of IPv6, along with new trends such as ubiquitous and mobile computing, is likely to make the number of potential destinations explode, thus exacerbating this fundamental scalability issue. In addition, there are some other contexts where the scalability of routing can be an important concern, such as large sensor networks in which the nodes have only a very limited amount of memory.

There is a fundamental relationship between the size of the state required by a routing algorithm and the quality of the routes that it can find. It is well-known that to accomplish shortest path routing on any network of n nodes, the routing table of each node needs to grow as $O(n)$. Indeed, if we denote by *path stretch* the ratio of the path length achieved by a routing protocol, divided by the shortest possible path on the graph, then it is known that any protocol that would keep the path stretch in the worst case strictly below three, would require a $O(n)$ bit state at each node as well [3]. As a direct consequence, if we want to significantly reduce the state required by routing algorithms in the future, we should consider algorithms that *may* inflate the path lengths.

One potential avenue is to design practical protocols that create, for all the nodes of the network topology, some virtual coordinates in a metric space such that the relative position of the nodes can be expressed as a function of their distance. Greedy forwarding consists in forwarding a packet to a node's neighbor closest to the destination. As this forwarding depends only on the distances between the neighbors of a node and the destination, it is a purely local mechanism. Further, the routing table consists only of the coordinates of a node's neighbors: This information scales as the maximum degree of the graph times the size of the coordinates. These are typically of the order of $O(\log(n))$, making these so called *geographic* (or *geometric*) routing schemes very scalable ($\log(n)$ bits are already required to merely name each node in the network). In addition, as the routing decision is a simple comparison of the relative distance between a set of neighbors and a destination, the forwarding decisions are fast and easy to implement.

In his famous 1967 *small-world* experiment [4], Milgram observes that human beings have the ability to efficiently route messages among themselves without having a full view of the topology; by just forwarding the messages to their acquaintances that they think are *the closest to the final destination*. To some extent, the Internet and a large category of random graphs exhibit similar small-world properties [5]. It is therefore natural to ask whether a more formal and explicit notion of distance can be obtained in the context of computer networks, that fits well the structure of such graphs.

Let $G = (V, E)$ denote the graph defined by the topology of the communication network. V represents the set of nodes (routers) and E denotes the set of bi-directional links connecting these nodes. Also, consider an embedding space (X, d),

[1] This work has been previously published in [1]. The present document contains an additional optional mechanism, presented in Section III-D, to further improve performance by using route asymmetry. It also contains new simulation results.

that is the metric space X equipped with the distance d.

For each node $v \in V$, define its set of neighbors $\mathcal{N}_v$, namely: $\mathcal{N}_v = \{w \in V, (v,w) \in E\}$. We recall the definition of a greedy embedding [6]:

Definition 1.1: A greedy embedding is a mapping $f : V \to X$ such that $\forall u, w \in V, u \neq w$:

$$\exists v \in \mathcal{N}_u \text{ such that } d(f(v), f(w)) < d(f(u), f(w)). \quad (1)$$

Applied to routing, this simply states that, if the node u is trying to send or relay a packet to the destination w, it will always find a neighbor v such that v is closer to w than u is, and thus that delivering the packet to v brings it closer to, and eventually at, its destination. Most geographical coordinate systems, including some virtual coordinate embeddings, do not produce greedy embeddings and require mechanisms to recover from local minima.

There is much theoretical work (some of which we describe in Section II) that considers whether a topology can be greedily embedded in a space (X, d), and under which conditions. Most of this work focuses on providing guarantees, and does not lend itself to implementation, as a full view of the topology is essential to most results. As a consequence, to our knowledge there exists no routing scheme that is practical, scalable (i.e., requiring an amount of memory polylogarithmic in n), achieves close to optimal path stretch and guarantees the success of routing. Our intent is to present such a scheme.

Outline: In the next section, we summarize the related work. In Section III, we present PIE and the embedding protocol. In Section IV, we provide an analysis of PIE. In Section V, we present an evaluation of the performances of PIE. We discuss practical relevance for Internet routing in Section VI and we finally conclude in Section VII.

II. Related Work

The idea of using coordinates for routing has been introduced in the context of wireless ad-hoc networks. In particular, the idea of using virtual coordinates (instead of the actual physical positions of the nodes) has been proposed as a mean to perform greedy routing without the need for a GPS receiver. [7], [8] and many others build practical schemes to create synthetic coordinates from the underlying topology. These are distributed methods, and can be implemented. However, they do not apply to all graph topologies (typically only on planar graphs) and cannot guarantee the success of greedy forwarding; the packets can be trapped in local minima.

Solutions such as *face routing* have been proposed to guarantee the success of geographic routing when local minima are present, see for instance [9]. These methods apply greedy routing by default and use a recovery mechanism when the packet is trapped in a local minimum. These deterministic recovery mechanisms only guarantee success of routing when the dimensionality of the underlying space is no more than two [10]. In addition, backtracking out of local minima significantly inflates paths lengths and induce high congestion [11].

In order to obtain greedy embeddings, it is therefore appealing to consider spaces of more than two dimensions. The fundamental tradeoff is to find a space of concise dimensionality (to guarantee scalability) that suits the embedding of a graph in a way that preserves the distances among the vertices (for routing performances). There is an ample body of theoretical work on graph embedding onto low-dimensional spaces (see [12] and references therein). Maymounkov [13] shows that $\log(n)$ is the minimal dimension for a Euclidean space to construct a greedy embedding of an arbitrary graph. The author also demonstrates that it is enough for trees, but his theoretical result, unfortunately, cannot be translated into a practical algorithm.

For some categories of graphs, it is possible to perform the embedding in a two dimensional Euclidean space. Indeed, Papadimitriou et al. [6] famously conjectured that such a space could embed any planar triangulation, and [14] confirms the conjecture. However, $O(n)$ bits are required to differentiate the points in the coordinate space.

Kleinberg [15] and Cvetkovski et al. [16] consider hyperbolic spaces of 2 dimensions and [15] demonstrates how to greedily embed any tree. However, here again the schemes results in coordinates of size $O(n)$ bits, and do not produce a significant gain in scalability. Very recently, Papadopoulos et al. [17] observed that uniform repartition of nodes onto a hyperbolic plane produces scale-free (Internet-like) graphs, and that the corresponding coordinates in the hyperbolic plane have desirable properties for greedy routing in these graphs. The reverse procedure has been used in [18] to find the hyperbolic coordinates of the Internet ASs that fit the actual AS topology as well as possible. Although this work gives precious insights to understand the relations between scale-free graphs and the hyperbolic space, it yields an embedding that is not greedy and it does not provide 100% packet delivery: routing may fail. PIE pursues similar goals but takes a different approach, it does not try to fit the coordinates to a predetermined space, but lets the embedding space be determined by the topology, using only local communications between the nodes.

[19] constructs a fully distributed practical embedding by projecting an n-dimensional graph topology onto a $O(\log(n))$ dimension Euclidean space using the Johnson-Lindenstrauss lemma. Despite attempting to preserve the relative distance between points, this method is *quasi*-greedy and introduces some distortion in the embedded topology, which creates local minima. It therefore requires a recovery mechanism that significantly increases the path stretch.

Gupta et al. [20] and Flury et al. [21] find a bounded stretch of 3 with $O(\log^2(n))$ coordinates for planar graphs [20] and combinatorial unit disk graphs [21]. For arbitrary graphs, the scheme of [21] also provides a stretch of $O(\log(n))$. However, these algorithms require a full, centralized knowledge of the topology in input.

The idea of trading off path stretch for routing table size is the core component of the work on *compact routing* (see for instance [22]). In [23], Thorup et al. show that it is possible to guarantee a path stretch no larger than three with routing tables of size $O(\sqrt{n \log(n)})$. Such compact routing schemes

have been successfully implemented in practice [24], [25]. We explore a different point in the tradeoff space, specifically, we relax the worst-case path stretch guarantee in order to provide polylogarithmic scalability, which is obviously needed to sustain any exponential growth of the Internet. We show in our evaluations that the relaxation of this guarantee does not disadvantage PIE in any way: it achieves significantly lower stretch than compact routing, and never higher than three.

[26] adapts the scheme of Thorup et al. for power-law graphs and obtains better scalability for the routing state, although still a fractional power of n.

[27] proposes a specialized scheme for power-law graphs, which provides polylogarithmic scalability, as PIE does. However, their method here again requires the complete topology graph in input and does not translate to a distributed protocol to build the routing tables. In addition, it relies on *tree routing*, that is, it uses only links that are spanned by some pre-constructed trees and neglect the others. PIE also constructs trees, but its geometric nature allows it to use all the links of the graph. We show in Section V that PIE finds shorter routes.

Distributed Hash Tables (DHTs) have been used to improve the scalability of routing as well (for instance, VRR [28]). However, such DHTs map to source routes that require $O(\sqrt{n})$ bits to be stored on many topologies, and $O(n)$ in the worst case. [29] and references therein use Delaunay triangulations to enable greedy forwarding with bounded stretch. However, unlike our work, they assume that the nodes exist in a Euclidean space. We assume nodes in an arbitrary connectivity graph. In particular, it has been shown that Euclidean spaces are not well suited to represent Internet nodes [30].

	[27]	[23], [24], [25]	[18]	[19]	PIE
polylogarithmic scalability	✓	×	✓	✓	✓
100% success rate	✓	✓	×	✓	✓
no recovery mechanism	✓	✓	✓	×	✓
distributed protocol	×	✓	✓	✓	✓

TABLE I
COMPARISON OF PIE WITH RELATED STATE-OF-THE-ART.

III. DESCRIPTION OF PIE

A. Model and Background

We consider a weighted graph $G = (V, E)$ associated with a function $w : E \to \mathbb{R}_+^*$ assigning a cost to each edge of G. w defines the usual (weighted) shortest path distance in G, that we denote by d_G. If $f : V \to X$ is an embedding of G into a metric space (X, d), f is said to have distortion D if:

$$\exists\, r > 0 \text{ such that } \forall\ u, v \in V,$$

$$r \cdot d_G(u, v) \leq d(f(u), f(v)) \leq D \cdot r \cdot d_G(u, v)$$

An embedding with distortion 1 is said to be isometric.

We are interested in situations where the host metric space is a standard k-dimensional metric space (X, d), where $X \in \mathbb{R}^k$, equipped with a l_p-norm such that $d(x, y) = \|x - y\|_p$ for all $x, y \in X$ and

$$\|x\|_p = \begin{cases} \sqrt[p]{\sum_{i=1}^{k} |x_i|^p} & \text{if } 1 \leq p < \infty, \\ \max_i |x_i| & \text{if } p = \infty, \end{cases}$$

for all $x \in X$. We denote by l_p^k such a space, and thus l_2^k denotes the usual k-dimensional Euclidean space.

As there is exactly one path between any two nodes in a tree, an isometric embedding of a tree is also greedy. Further, it is known (see Linial et al. [31], Theorem 5.3) that a tree can be isometrically embedded in $l_\infty^{O(\log n)}$. Given these two pieces of information, we could imagine a routing scheme that first extracts a tree T spanning the connection graph G, embeds it isometrically in $l_\infty^{O(\log n)}$ and uses the resulting greedy embedding of G to perform greedy routing. However, this approach would not work in practice, for two main reasons: First, the isometric tree embedding algorithm proposed in [31] requires a full, centralized knowledge of the tree, as it recursively divides it in balanced subtrees. Second, routing over a tree is clearly inefficient because a significant number of links may not be taken into account, possibly leading to poor performance in terms of path stretch and congestion.

In the following, we address these two problems. By relaxing the *deterministic* guarantee on the dimensionality, we are able to devise a different, simple, isometric embedding algorithm that does not need global knowledge of the topology and is easy to implement in a distributed scenario. As shown in Section IV, the guarantee on the dimensionality becomes $O(\log^2 n)$ *with probability one* (almost surely), on the relevant categories of random graphs.

The second problem due to tree routing is addressed by constructing multiple trees with different locality levels. In such a scenario, not all the trees would span the whole graph, but most would span only a local portion of it, according to their locality level. However, the union of all the trees at each locality level would cover the whole graph. As such, each node is covered by one tree for each locality level, that is, by $\log(n)$ trees in total if we choose $\log(n)$ locality levels.

Here are the high level steps of PIE:

- Extract several (rooted) trees with different locality levels from the graph, with at least one spanning the whole graph.
- Embed each of these trees in a separate coordinate system.
- When forwarding a packet, choose a tree on which to perform greedy routing and send the packet to the neighbor that provides the best progress towards the destination in this coordinate system.

In the next section, we present the distributed greedy embedding algorithm in detail, using one spanning tree. The extension to several trees is explained in Section III-C.

B. Isometric Tree Embedding

Let T denote a rooted spanning tree of G. We explain here how to embed T, and we provide later two distributed algorithms that (i) extract T out of G and (ii) embed T.

Let O be a node of the tree T. At the beginning of the algorithm, O is set to be the root of the tree, and the coordinate $\{0\}$ is assigned to it. Let us denote by S the set of children of O that consists of the nodes $S = \{v_0, v_1, \ldots, v_{s-1}\}$, where $s = |S|$ is the cardinality of S.

For each child $v_i \in S$, compute a binary representation of its index i. We denote by $b_i = \langle b_i^0, b_i^1, \ldots, b_i^{h-1} \rangle$ such a representation, where $h \leq \lceil \log_2(s) \rceil$.

Let A_i be the set formed by v_i along with all its descendants in T. The algorithm appends h new coordinates $\langle c_i^0, c_i^1, \ldots, c_i^{h-1} \rangle$ to all the vertices u in A_i as follows:

$$c_i^j = \begin{cases} -d_T(u, O) \text{ if } b_i^j = 0, \\ d_T(u, O) \text{ if } b_i^j = 1, \end{cases} \quad (2)$$

$0 \leq j \leq h-1$. After that, each node in S plays the role of O, and the algorithm repeats the same procedure. This way of assigning the coordinates goes from the root to the leaves in one pass and can be implemented in a way that induces only local communication between a node and its neighbors. In particular, at each step, the node O is higher in the tree than the nodes that receive the new coordinates, and Eq. (2) does not need to be evaluated for all the vertices in A_i at the same time. Each node can simply infer them based on the coordinates of its parent in the tree. Therefore, each node O needs only to transmit its own coordinates along with the binary map b_i to each of its children v_i.

The binary map b_i can be any variable length binary representation of i obtained with a prefix-free code. In particular, such a map of length $h \leq \lceil \log_2(s) \rceil$ can be obtained using a Huffman code to represent the s children of O when they are equiprobable.

The scheme can be slightly improved if we note that if a node O is not the root and $s = 1$ (i.e., it has only one child), assigning a new coordinate to all the descendants of O would have no effect on their relative distance under the l_∞-norm. In this case, the binary map does not need to be sent. A step-by-step example of the embedding is shown in Figure 1.

The greedy forwarding procedure is straightforward: When forwarding a packet, a node considers all its neighbors that are closer to the destination and chooses the one that minimizes the overall path length (i.e., taking into account the cost of the link to go to this neighbor). Specifically, a node v forwarding a packet to a destination t chooses the node that satisfies:

$$\underset{u \in \mathcal{N}_v \text{ s.t. } \|(t-u)\|_\infty < \|(t-v)\|_\infty}{\arg\min} \{w(v,u) + \|(t-u)\|_\infty\}.$$

Note that this forwarding procedure considers *all* the neighbors in G, and not only the neighbors in T. This enables shortcuts off the tree. Indeed, we prove in Section IV that this embedding of T yields a greedy embedding of G. Therefore, this forwarding procedure always returns a next-hop closer to the destination (except if v is already the destination).

Algorithm Specification: The overall algorithm proceeds in two steps. First, a spanning tree is extracted from the graph and then the virtual coordinates are computed based on this tree. We specify these two steps in the form of two distinct modules, the `tree_maintainer` and the `coordinates_maintainer`. The `tree_maintainer` implements a distributed spanning tree construction by using the well-known STP protocol [32]. Recall that in this protocol, each node chooses an ID and the node with the largest ID eventually becomes the root of the whole spanning tree. During our experiments on power-law graphs, we observed slightly better routing performances when the root was the highest degree node. We thus choose the ID to be the degree of the node (plus a small random salt to break ties if needed).

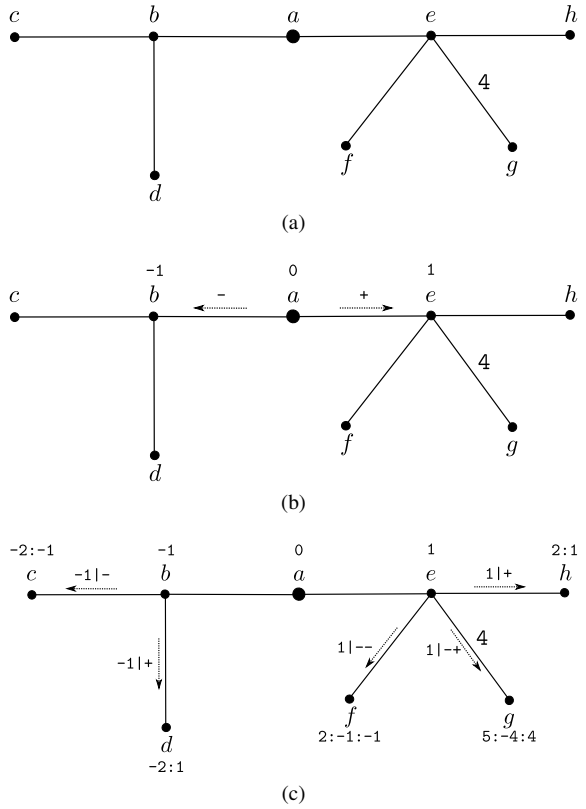

Fig. 1. Example of isometric embedding from the root to the leaves. (a) shows the tree to embed. The vertices are named a to h. The root is the node a. We assume that all the edges have a weight of 1, except the edge (e, g) that has a weight of 4. (b) After having picked the coordinate $\{0\}$, the root sends the binary map (here represented by + and −) corresponding to each of its children. In (c), the nodes b and e play the role of O and send their coordinates to each of their children, along with the corresponding binary maps. Note that the node e has 3 children, resulting in a map of 2 bits for two of them and one bit for the last one. The algorithm does not require all the nodes to have the same number of coordinates, the l_∞-norm is simply applied on the first common coordinates. For example, if one wants to compute the distance between the node d and the node g, the coordinates to use are $\{-2:1\}$ and $\{5:-4\}$. As $|-2-5| > |1+4|$, the distance is $|-2-5| = 7$.

The STP protocol has been augmented with a straightforward improvement, in order for each node to learn who its children are when it receives messages from its neighbors. The `coordinates_maintainer` acts separately but uses the `tree_maintainer` in order to access the list of children. In realistic scenarios, the topology may of course change over time and the links may be asynchronous. Therefore, these two modules typically act on a periodic basis in order to accommodate possible changes in the tree. The pseudo-codes corresponding to the distributed versions of the `tree_maintainer` and `coordinates_maintainer` are shown in Algorithms 1 and 2, respectively. We use the following conventions for the pseudo-code:

Algorithm 1 `tree_maintainer` at node u

Init:
$children \leftarrow \emptyset$
$u.rootId \leftarrow$ degree of u (+ random salt in $[0, 1[$ to break ties)
$u.height \leftarrow 0$
$u.parent \leftarrow \emptyset$

Periodically:
send treeMsg($u.rootId, u.height, u.parent$) to each neighbor of u

Upon reception of treeMsg msg from neighbor v:
$w_v \leftarrow w(u, v)$ ▷ cost from u to v
if $(msg.rootId > u.rootId)$ or $(msg.rootId = u.rootId$ and $msg.height + w_v < u.height)$ **then**
  $u.parent \leftarrow v$
  $u.height \leftarrow msg.height + w_v$
  $u.rootId \leftarrow msg.rootId$
end if
if $msg.parent = u$ and $v \notin children$ **then**
  $children$.add(v)
end if
if $msg.parent \neq u$ and $v \in children$ **then**
  $children$.remove(v)
end if

procedure GETCHILDREN
  return $children$
end procedure

Algorithm 2 `coordinates_maintainer` at node u

Init:
$u.coords[0] \leftarrow 0$

Periodically:
$children \leftarrow$ `tree_maintainer`.GETCHILDREN
if change in childhood **then**
  NOTIFYCHILDREN
end if

procedure NOTIFYCHILDREN
  $s \leftarrow$ number of children
  if $s = 1$ **then**
    send coordMsg($u.coords$) to $children[0]$
  end if
  if $s > 1$ **then**
    for $i = 0$ to $s - 1$ **do**
      $b_i \leftarrow$ prefix-free binary representation of i
      send coordMsg($u.coords, b_i$) to $children[i]$
    end for
  end if
end procedure

Upon reception of coordMsg msg from parent p:
$w_p \leftarrow w(u, p)$ ▷ cost from u to p
$l \leftarrow msg.coords$.length
for $k = 0$ to $l - 1$ **do**
  if $msg.coords[k] < 0$ **then**
    $u.coords[k] \leftarrow msg.coords[k] - w_p$
  else
    $u.coords[k] \leftarrow msg.coords[k] + w_p$
  end if
end for
for $k = 0$ to $msg.b$.length-1 **do**
  if $msg.b[k] = 0$ **then**
    $u.coords[l + k] \leftarrow -1 \cdot w_p$
  end if
  if $msg.b[k] = 1$ **then**
    $u.coords[l + k] \leftarrow w_p$
  end if
end for
if $u.coords$ changed **then**
  NOTIFYCHILDREN
end if

- $a.b$ denotes the field b of the element a.
- $A[i]$ denotes the element at index i of the data structure A (the indexes start at 0).
- A.length denotes the number of elements in the data structure A.
- treeMsg and coordMsg are the two message types used by the `tree_maintainer` and the `coordinates_maintainer`, respectively. When they are constructed, they receive as arguments the values of the fields that they will carry.

The other notations should be clear from the context.

C. Extension to Several Trees

This embedding is a significant improvement over tree routing. It can still be improved by using multiple trees. Building only one tree spanning the graph takes into account exactly $(n - 1)$ links when computing the coordinates and it ignores all the other links. This can lead to some sub-optimal routing decisions when the shortest path between two nodes contains a link that is not included in the spanning tree.

We now describe how to use several trees so that any given link has a high probability of being spanned by one of the trees. An obvious solution would be to construct multiple spanning trees. However, in order to keep a small overall number of coordinates, each node has to belong to a small number of trees. If all these trees span the whole graph and have randomly chosen roots, it is likely that some of these roots will be close to each other; this would lead to similar, redundant trees, with little or no performance gain.

Instead, we propose to partition the graph m times: The first partition divides the graph in two pieces, the second partition divides the graph in four pieces and, more generally, the l-th partition divides the graph in 2^l pieces. Each of these m partitions defines what we denote a *locality level*. For the locality level l, each of the corresponding 2^l pieces of the graph is spanned by one tree and we denote these 2^l trees the trees *of level* l. Note that there is only one tree of level 0 and that it spans the whole graph. Of course, computing such exact partitions for each of the m locality levels would require a global knowledge of the graph.

In order to keep a distributed solution, we slightly modify the procedure and adopt the following election process: For each locality level $0 \leq l \leq m - 1$, each node elects itself as the root of a tree of level l with a probability of $2^l/n$, independently of the other nodes. We have therefore an *expected* number of 2^l trees of level l, for all $0 \leq l \leq m - 1$. Each of the trees is then constructed in a similar way as described in the previous section; for every locality level l, each node chooses to belong to the tree of level l whose root

is the closest (breaking ties arbitrarily). Each node maintains therefore m independent sets of coordinates, corresponding to the m trees (of levels 0 to $(m-1)$) to which it belongs. We will see in Section V that taking $m \in O(\log(n))$ leads to substantial performance gain and makes the performances of the scheme to scale with the size of the network.

The only necessary condition to route all the packets successfully is that all the nodes have at least one of their m sets of coordinates in common (i.e., that they belong to at least one common tree). This condition can be trivially satisfied by ensuring that at least one node deterministically becomes the root of a tree of level 0. This node can be for instance the one having the largest ID, as in the single-tree case. Now, when evaluating the distance between two nodes, one just chooses the l_∞-norm that is minimized over the trees that the two nodes have in common. We denote by u_l the coordinates of node u in the tree of level l to which it belongs. In addition, we denote by $T^l_{u,t}$ a tree of level l to which both the node u and the node t belong. When a node v wants to transmit a packet to a destination t, it chooses the node that satisfies:

$$\arg\min_{u \in \mathcal{N}_v} \min_{l \,:\, \exists T^l_{u,t} \text{ s.t. } \|(t_l - u_l)\|_\infty < \|(t_l - v_l)\|_\infty} \{w(v,u) + \|(t_l - u_l)\|_\infty\}. \quad (3)$$

This forwarding procedure needs to be able to uniquely identify the trees. This can easily be done using the identifier of the root, of size $\log(n)$. Figure 2 shows an example of our embedding using several trees.

The intuition now is that a large number of small trees having a high locality level provides fine-grained coordinates for local paths, while larger trees of lower levels provide coarse-grained coordinates for the longer routes and tie everything together, much like in a divide-and-conquer strategy.

Algorithm Specification: The distributed implementation simply consists in a generalization of the `tree_maintainer` and the `coordinates_maintainer` to use m independent trees, as described above.

D. Source-aided Geometric Routing

The routes found by PIE are not necessarily symmetric: When a source s sends a packet to a destination d, the set of edges chosen by the forwarding procedure may be different than if the packet were sent by d to s. To see this, consider again the example of Figure 2b. The path from s to d goes through v, whereas the path from d to s goes through u and is one hop longer. Let $P_{s\to d}$ denote the path found by PIE when routing from s to d. It consists in the sequence of nodes through which any packet from s to d is routed using the geometric coordinates, including s and d themselves. Similarly, let $P_{d\to s}$ denote the path from d to s. For a path P, its length can be written as $d(P) = \sum_{i=1}^{|P|-1} w(P_i, P_{i+1})$, where P_i is the i-th element of the path P and w is the link weight function. Note that the fact that the routes found by PIE are not necessarily symmetric implies that $d(P_{s\to d})$ is not necessarily equal to $d(P_{d\to s})$. This motivates the following

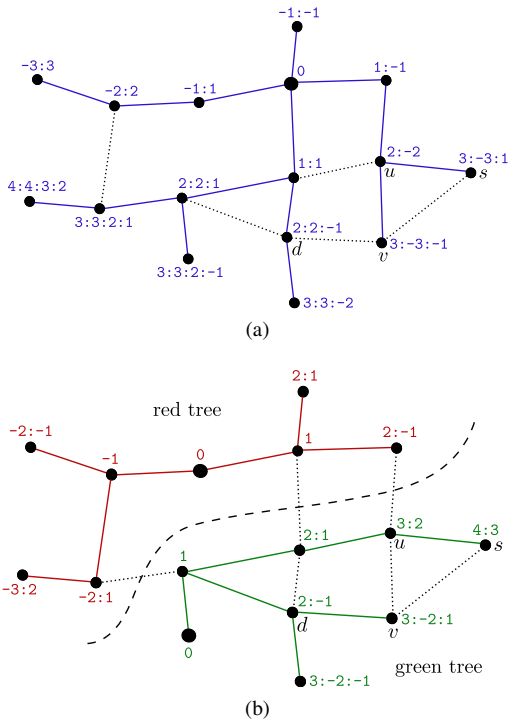

Fig. 2. Example of an embedding using several trees. We assume for clarity that all the edges have a cost equal to one. (a) shows a tree of level 0 that spans the whole graph (with solid edges) and a corresponding set of coordinates at each node. (b) shows two trees of level 1, each with solid edges, along with the corresponding coordinates, which we denote by "red" and "green". We are interested in the case where a source s wants to send a message to a destination d. s will compare the coordinates in the trees that its neighbors have in common with d. Here, the neighbors of s are u and v and they both have the level 0 and the "green" level 1 sets of coordinates in common with d. s will find that u is at distance 4 of d using the level 0 coordinates, and at distance 3 using the "green" coordinates. Similarly, v is at distance 5 with the level 0 coordinates and 1 with the "green" ones. Therefore, s will forward the packet to v, which is the optimal choice here. Note that if only the tree of level 0 was present, s would have forwarded the packet to u. The "green" tree provides a valuable shortcut in this situation.

optional source-routing extension of PIE in order to use the shortest of both routes.

Assume that two nodes s and d are engaged in a bi-directional communication, which involves packets sent by s to d as well as packets sent by d to s, as would for instance happen with a TCP connection. Assume also, without loss of generality, that s sends the first packet. Our approach relies on identifying the bifurcations between $P_{s\to d}$ and $P_{d\to s}$ when the first packet in each direction is sent. Consider an intermediate node v of the return path $P_{d\to s}$, and let u be the node that precedes v in $P_{d\to s}$ (see Figure 3). The first packet sent by d to s goes through u and then through v. When v receives this packet, it checks whether u would be the next hop for the destination d using the virtual coordinates. If this is not the case, there is a bifurcation and the ID of u is added to the packet's header. The first two packets in each direction are also used to measure $d(P_{d\to s})$ and $d(P_{s\to d})$.

When s receives the first packet back from d, it compares the length of the two paths. If $d(P_{d\to s}) < d(P_{s\to d})$, the reverse path is shorter and s stores the bifurcation(s) – namely, a sequence of node IDs – from the header of the packet that

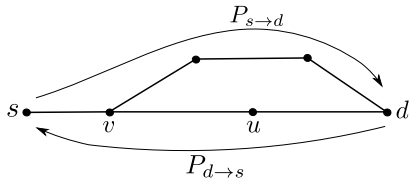

Fig. 3. Illustration of route asymmetry.

Algorithm 3 Forwarding procedure at node $v \in P_{s\to d}$

Upon reception of first packet pkt **with source** s **and destination** d**:**
Let u be the predecessor of v in $P_{s\to d}$
$pkt.dist1 \leftarrow pkt.dist1 + w(u,v)$
if u does not satisfy Expression (3) for destination s **then**
  $pkt.bifSet \leftarrow pkt.bifSet \cup u$
end if
if $v = d$ **then**
  Store $d(P_{s\to d}) \leftarrow pkt.dist1$
  if $pkt.dist2 \neq$ `null` **then**
    Store $d(P_{d\to s}) \leftarrow pkt.dist2$
  end if
  Store $bifSet_{s\to d} \leftarrow pkt.bifSet$
end if

When forwarding a subsequent packet pkt **from** s **to** d**:**
if $pkt.bifSet \cap v.neighbors \neq \emptyset$ **then**
  Forward to the (first) node in $pkt.bifSet \cap v.neighbors$
else
  Forward using Expression (3)
end if

Algorithm 4 Packet emission procedure at source s for destination d

When sending the first packet pkt **to** d**:**
set $pkt.dist1 \leftarrow 0$
if $d(P_{d\to s})$ is known **then**
  set $pkt.dist2 \leftarrow d(P_{d\to s})$
else
  set $pkt.dist2 \leftarrow$ `null`
end if

When sending a subsequent packet pkt**:**
if $d(P_{d\to s}) < d(P_{s\to d})$ **then**
  set $pkt.bifSet \leftarrow bifSet_{d\to s}$
end if

it just received from d. These bifurcations are then added to each packet that s sends to d. Now, when receiving a packet from s to d, each intermediate node in $P_{s\to d}$ checks if one of its neighbors belongs to the bifurcation set indicated in the packet. If a neighbor belongs to the set, the packet is forwarded to this node. Otherwise, the virtual coordinates are used. The complete procedure at the intermediate nodes is described in Algorithm 3 and the procedure at the source s is described in Algorithm 4.

In the pseudo-code, a packet pkt may contain a bifurcation set $pkt.bifSet$. This set is built during the first bi-directional exchange, and it may be used by the source s if the reverse path is shorter than the direct one. The variables $pkt.dist1$ and $pkt.dist2$ are used to measure $d(P_{s\to d})$ and $d(P_{d\to s})$, respectively.

Note that identifying the first packet in each direction only requires only one additional bit in the header. Storing the set of bifurcations may require more memory. However, we observe in Section V that the number of node IDs that compose the bifurcation set is below 1.5 on average, at most 6, and essentially constant with the network size.

IV. Analysis

In this section, we derive some simple facts about the scalability and performance of PIE when applied on Internet-like graphs. For simplicity, we consider only the embedding of a single spanning tree T on an unweighted graph (that is, using the hop count metric). Extensions of the results to multiple trees and weighted graphs are immediate in most cases.

A. Internet-like graphs

Let us write $\mathrm{diam}(G)$ for the diameter of a graph $G = (V, E)$. For a node $u \in V$, we write $\dim(u)$ for the number of coordinates assigned to u by the single tree embedding procedure of PIE. Let us also denote by $\dim(G)$ the highest such number, i.e., $\dim(G) = \max_{u\in V} \dim(u)$.

It has been pointed out by several research groups (see [33], [34]) that the connectivity graph of the Internet exhibits a power law node-degree distribution, both at the router and at the AS levels. In these graphs, the proportion of nodes having degree k is proportional to $k^{-\lambda}$ for some constant $\lambda > 1$. For the Internet, λ has consistently been estimated in the range $2 < \lambda < 2.3$ [5]. Such a degree distribution (in particular when $2 < \lambda < 3$) leads to very particular structural properties, among which the fact that the graphs typically exhibit extremely small distances between the vertices (with $\mathrm{diam}(G) \sim \log n$ [35]), hence the *small-world* denomination. In the following, we denote by $G(n, \lambda)$ the realization of an n-nodes random graph such that the expected degree sequence $(k_1, k_2, ..., k_n)$ follows a power law with exponent λ and an edge between two nodes u_i and u_j is created independently with probability proportional to $k_i k_j$ [35]. Some authors use the term *scale-free* for such graphs. As this term is not defined unambiguously and may imply some other properties that we do not need here [36], we only use the term *power law graph*.

B. Success Ratio

Theorem 4.1: For any connected graph G, the embedding of G produced by PIE ensures the success of routing.

Proof: We need to show that PIE produces a greedy embedding. As T is a subgraph of G that contains all the vertices of G, it is clear that a greedy embedding of T is also a greedy embedding of G. In addition, an isometric embedding of T is a greedy embedding of T. It suffices therefore to show that PIE produces an isometric embedding of T. For any node $u \in T$, write $\langle u^0, u^1, \ldots, u^{\dim(u)-1}\rangle$ its coordinates. For any two nodes $u, v \in T$, write $O_{u,v}$ their least common ancestor in T. Every node above $O_{u,v}$ in T assigns the same coordinates to u and v. $O_{u,v}$ assigns coordinates with magnitude $|d_T(O_{u,v}, u)|$ to u and $|d_T(O_{u,v}, v)|$ to v in Eq. (2), with at least two of these

coordinates, say u^h and v^h, having opposite signs. Therefore, $\exists h$ s.t. $|u^h - v^h| = d_T(O_{u,v}, u) + d_T(O_{u,v}, v) = d_T(u, v)$, because $O_{u,v}$ is the least common ancestor of u and v. Moreover, as every node below $O_{u,v}$ in T assigns coordinates with a magnitude strictly smaller than $|d_T(O_{u,v}, u)|$ to u and $|d_T(O_{u,v}, v)|$ to v, $\|u - v\|_\infty = |u^h - v^h| = d_T(u, v)$. ∎

C. Scalability

We give a probabilistic upper bound on the number of coordinates that are required to describe the position of the nodes in large graphs.

Theorem 4.2: Let $G(n, \lambda)$ be an n-nodes realization of a power law graph with $2 < \lambda < 3$. We have:

$$\dim(G(n, \lambda)) \in O(\log^2(n)) \tag{4}$$

almost surely.

Proof: Let r denote the root of T. For any node u, let P be the set of all the nodes above u in the unique path from r to u in T. For each node $v \in P$, the embedding algorithm assigns $\lceil \log_2(\delta_v) \rceil$ new coordinates to u, where δ_v denotes the degree of the node v (see Eq. (2)). Obviously, we have that $\forall v \in G, \delta_v \leq \Delta$, where Δ is the maximum node degree in G. In addition, as T is the union of the shortest paths from r to all the other nodes in G, we have $|P| = d_G(r, u)$. We can therefore write the upper bound $\dim(u) \leq \lceil \log_2(\Delta) \rceil d_G(u, r)$. We have:

- $\Delta < n$,
- $d_G(u, r) \leq \text{diam}(G) \in O(\log n)$ a.s. ([35] Theorem 4).

Relation (4) follows. ∎

This means that, with probability one, PIE embeds T (and thus G) in $l_\infty^{O(\log^2(n))}$. If we consider the multi-tree case, each node belongs to $O(\log n)$ trees and thus PIE almost surely embeds G in $l_\infty^{O(\log^3(n))}$. Note that this bound holds for any graph with diameter $O(\log n)$. In particular, it holds for more classic random graphs (see for example [37]).

D. Performance

As a node "knows" all of its neighbors, the algorithm finds all the 1-hop routes with stretch 1. Therefore, a route between a source u and a destination v may exhibit a stretch larger than 1 only if $d_G(u, v) \geq 2$.

As the embedding is greedy, the longest possible route that the routing procedure can find has length $\text{diam}(T) \leq 2 \cdot \text{diam}(G)$. Therefore, the worst possible stretch is $\text{diam}(G)$. Using Theorem 4 in [35], we have just shown the following:

Theorem 4.3: Consider $G(n, \lambda)$ an n-nodes power law graph with $2 < \lambda < 3$. The worst case stretch over all node pairs in $G(n, \lambda)$ of a route found by PIE is $O(\log n)$ a.s.

Note that this is a worst-case bound when only one tree is used. We observe in the next section that both the average and the maximum stretch do not vary with n.

E. Protocol Overhead

We provide a few key observations related to the protocol overhead:

- Maintaining a tree requires that each node maintains a shortest path to that tree's root. If $\log n$ trees are used, $\log n$ such shortest paths need to be maintained.
- As a comparison, shortest paths algorithms typically rely on distributed protocols that are extremely similar to the spanning tree construction, building n spanning trees, one rooted at each node.
- All the control messages used by PIE have a size poly-logarithmic in n.
- More than network overhead, the re-computation of the routing table is perhaps the biggest burden of traditional algorithms. PIE only manipulates extremely small (poly-logarithmic) routing state and removes this issue.
- For any pro-active routing protocol used in a dynamic topology, there exists a necessary tradeoff between the frequency with which control messages are sent, and the ability of the algorithm to successfully bring packets at destination at any time. We observe in the next section that, due to its geometrical nature, PIE is significantly more resilient to network failures than standard algorithms, and requires to re-compute its state less often.

V. Evaluation

A. Settings

We evaluate the behavior of PIE by using simulations on several topologies. We wrote our own simulator that we optimized to simulate routing on large graphs. We performed extensive simulations of PIE on several Internet-like graphs [38], [39], [40], [41], [42], on which the results were very similar. To spare space and to be able to explore more of the parameter space, we display results only for the two following topologies:

- DIMES [38] is a collaborative project that uses thousands of end-host agents to reproduce the topology of the Internet as accurately as possible. We use the AS-level dataset of March 2010. We consider all the links as symmetrical and remove the nodes that are not part of the main component, yielding a topology graph of 26,424 AS nodes. We measured λ to be about 2.06 for this graph.
- GLP (Generalized Linear Preference) [39] is a preferential attachment model that builds on the well-known scheme of Barabási et al. [40]. This model allows us to tune λ while producing graphs that exhibit some given properties such as characteristic path length, clustering coefficient or distribution of the highest degrees. The main benefit, of such a synthetic model over a fixed snapshot of the current Internet, is that it allows to generate larger graphs of varying size in order to study the scalability of PIE.

We consider weighted and unweighted graphs. The weights are drawn uniformly in [1,10], which can for instance be thought of as a function of a financial cost and a link capacity, and are comparable to the ISP's link weights range. Such a cost

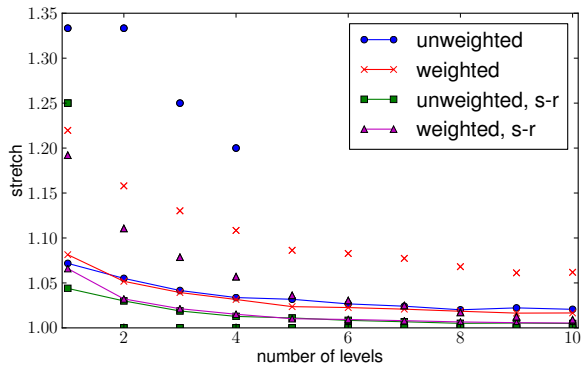

Fig. 5. DIMES topology. Stretch as a function of the number of levels m, with and without the optional source-routing ("s-r") mechanism. The mean and 90-th percentile are shown.

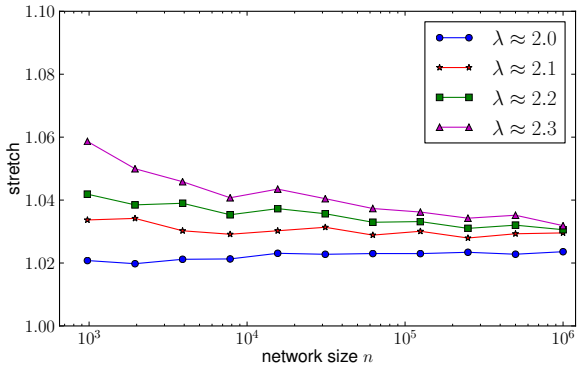

Fig. 6. GLP topology. No source routing. Average stretch as a function of n, for $2 \leq \lambda \leq 2.3$. For each value of n, the 95-th percentile of the stretch was $4/3$ or less.

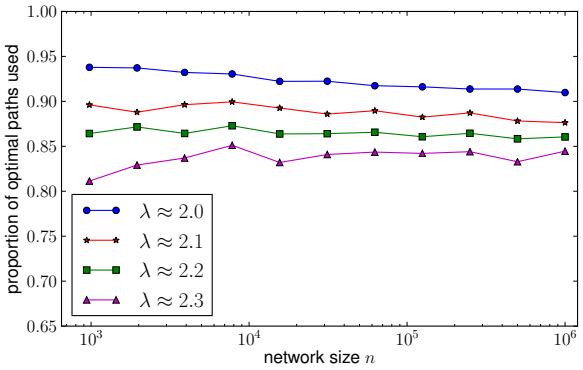

Fig. 7. GLP topology. No source routing. Proportion of shortest paths among all the paths found by PIE, as a function of n, for $2 \leq \lambda \leq 2.3$.

function naturally produces a large amount of violations of the triangle inequality in the graph, which are known to induce much distortion in Euclidean embeddings [30]. For each setting, the statistics have been obtained by simulating routes between 10^5 distinct, randomly chosen source-destination pairs, over 10 independent experiments using different seeds.

B. Results

1) Performance: Figure 4 shows the CDFs of the path stretchs obtained on the DIMES topology, and Figure 5 shows the average stretch as a function of the number of locality levels. On both figures, the results are shown both with and without the source-routing extension introduced in Section III-D. Even when this extension is not used, the results are excellent: when only two trees are used, more than 97.5% of the routes have stretch below 1.3. For $m \geq 4$, the average stretch is below 1.035 and, on the unweighted graph, 90% or more of the routes found by PIE are the shortest. For $m \geq 8$, the average stretch is below 1.023. The maximum stretch observed over all the simulations on the unweighted graph was 2, which is indeed better than the best possible upper bound of 3 for compact routing schemes. The source-routing extension further reduces the stretch: with this mechanism enabled, the average stretch is below 1.007 for both the weighted and unweighted graphs with $m \geq 8$. Unless otherwise specified, we present the remaining results *without* the source-routing extension.

Figure 6 shows the evolution of the average stretch when the network grows, on unweighted GLP graphs with $2 \leq \lambda \leq 2.3$. Here and in the following experiments, the number of levels is $m \in O(\log_2 n)$[2]. Figure 7 shows the proportion of shortest paths found by PIE. It appears clearly that the good quality of the routes found by PIE scales perfectly with the size of the network: the average stretch always stays below 1.06 and the proportion of shortest routes above 80%. The stretch even appears to slightly decrease with n, this comes from our choice for the computation of m.

[2]The exact function that we use is $m = \lfloor \log_2(n) - 7 \rfloor$ (this function yields $m \in \{2, \ldots, 12\}$ for the values of n that we consider).

2) Scalability: Figure 8 shows the total number of coordinates required at each node by all the trees in the last scenario. Also plotted is a (shifted) fit with a function $O(\log^3 n)$ (note the logarithmic scale of the x-axis). As predicted in Section IV-C, the embedding of m trees by PIE produces $O(\log^3 n)$ coordinates, hence meeting the scalability promises.

For the optional source-routing extension, Figure 9 shows the average and maximum number of bifurcations that need to be included in the packets' headers. We compute the average only over the routes that do benefit from using such a bifurcation set (it would be lower otherwise). Both the average and maximum sizes remain very low – always 6 node IDs or less – and, importantly, they do not grow with the network size. This essentially means that the source-routing extension only incurs a constant overhead in the packets' headers, and the benefits provided in term of routing stretch do not incur a scalability penalty.

3) Resilience to Network Failures: We consider a scenario in which some randomly chosen nodes fail. If a node has failed, its neighbors cannot send messages to it anymore. We evaluate the success ratio of the routing procedure, after some proportion of the nodes has failed, but before the algorithm has had time to react and adapt the routing tables.

Figure 10 shows the proportion of successful routes (success ratio) as a function of the percentage of nodes that have failed, on the unweighted DIMES topology. PIE maintains a

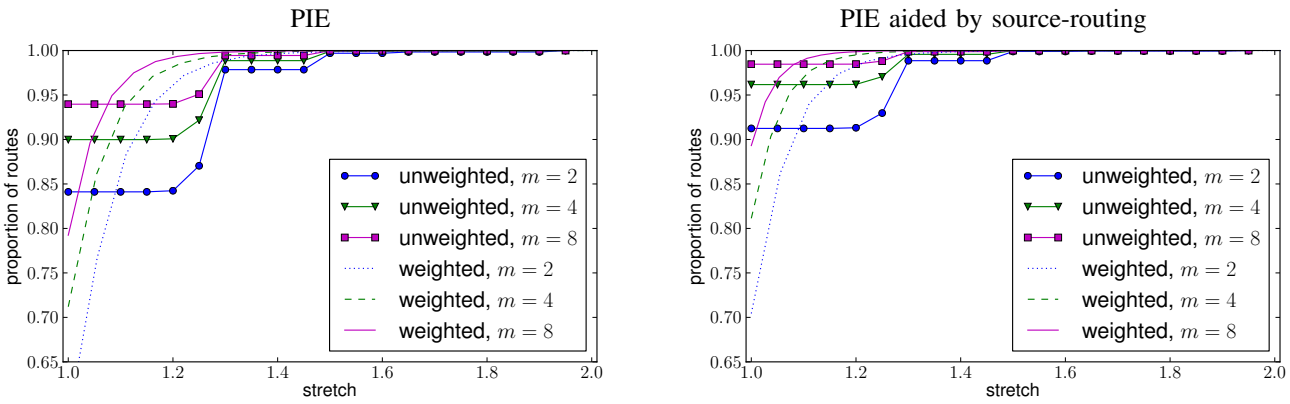

Fig. 4. DIMES topology. Empirical CDFs of the path stretchs for several values of m (the number of levels), with and without costs attributed to links. Left: results for PIE alone. Right: results for PIE aided by the optional source-routing mechanism.

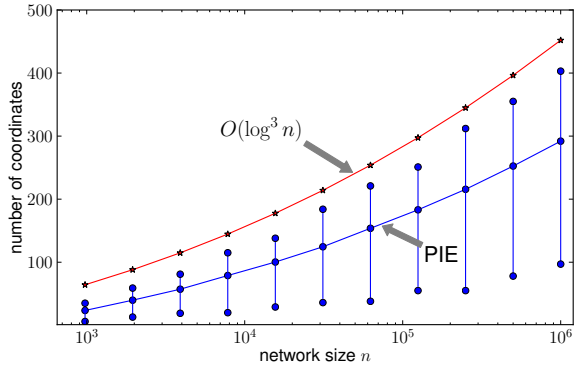

Fig. 8. GLP topology. Scalability of the total number of coordinates. The mininimum, maximum and average number of coordinates per node are shown. The curve above is a (shifted) plot of $(2 + \log_{10} n)^3 \in O(\log^3 n)$.

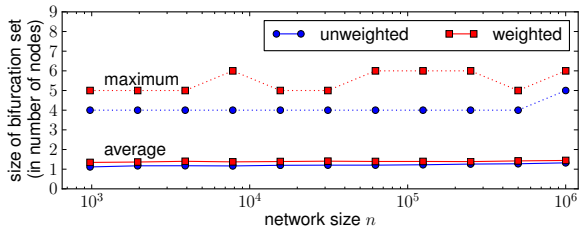

Fig. 9. GLP topology. Size of the bifurcation set used for the optional source-routing mechanism. The average and maximum over 10^5 routes are shown.

significantly higher number of successful paths than traditional shortest paths algorithms. This is explained by considering the greedy nature of the forwarding procedure of PIE: forwarding to *any* neighbor that is closer to the destination provides route diversity gain, while schemes producing 1-to-1 mappings between destinations and next hops (including compact routing schemes), do not benefit from this route diversity.

When only one tree is used, PIE consistently reduces the number of routing failures by at least 20%, and this proportion jumps to 50% when $m = 8$. Even in the extremely unlikely scenario where 10% of the Internet fails, PIE could manage to maintain a high success ratio, about 90%. One direct consequence is that, for a given success ratio, PIE does not need to recompute its state as often as standard algorithms.

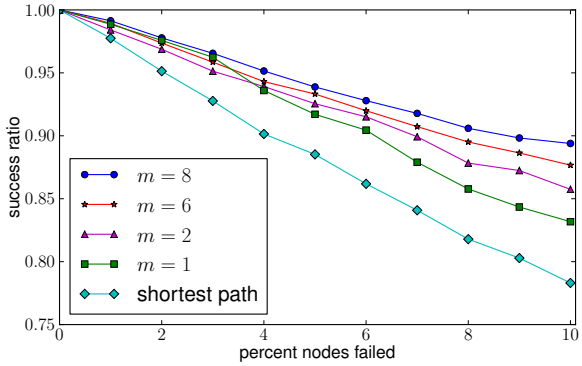

Fig. 10. DIMES topology. Proportion of successful routes as a function of the percentage of failed nodes, for several values of m.

4) Comparison with the State of the Art: We compare PIE with the *general* compact routing scheme [23] (that we denote by TZ). In addition we also compare it with the *specialized* compact routing scheme [27], that is especially targeted for power law graphs (that we denote by BC). It is proposed in [27] to combine TZ and BC in order to obtain a new scheme (that we denote by TZ+BC), which uses the best of the routes found by TZ and BC taken together. We recall that TZ achieves only $O(\sqrt{n \log n})$ scalability for the routing table size, and BC requires a complete knowledge of the graph at all the nodes and is not translated in a distributed protocol. TZ+BC accumulates these two fundamental issues.

The authors of [27] publicly provide the graphs that they used to obtain their simulation results with $\lambda \in \{2, 2.1, 2.2\}$. We can therefore run PIE (with and without the optional source-routing extension proposed in Section III-D) on these exact same graphs and compare the results. This is shown on Figure 11. Critically, PIE performs significantly better than its less scalable, respectively centralized, counterparts. It even finds similar or better routes than the best routes found by TZ and BC taken together.

VI. Discussion

While we demonstrate the scalability of our routing method from a theoretical point of view and provide the corresponding distributed protocol, translating this protocol to a deployment

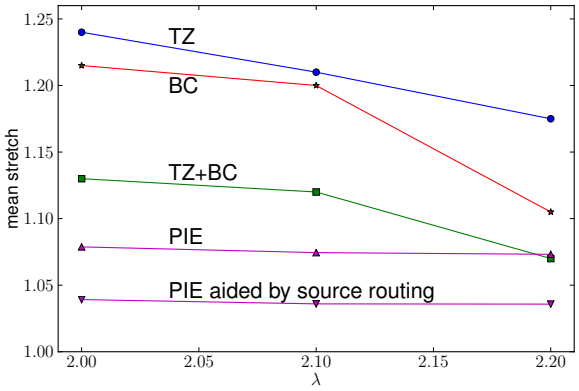

Fig. 11. Comparison of the mean stretch obtained by PIE, [23] (TZ) and [27] (BC). The values plotted for BC and TZ come from [27], as do the graphs used for the simulations. In this scenario, the number of nodes is 10^4, but the main connected components of the graphs have size $n \approx 8400$.

environment requires a couple more steps. In a single administrative domain, its deployment would be easy, as ASs run their own routing protocol internally. Since some ASs are relatively large, they would benefit from the scalability of our scheme. Similarly, our protocol would be practical over large overlay networks, where the weights would be dependent on the target that the overlay aims to achieve (for instance, minimize delay between overlay nodes).

For the wider Internet, the issue becomes to integrate our protocol with BGP. The simplest integration would be to build tree(s) of level 0 between the ASs and trees of higher levels within the ASs. Internally, the ease of geometric routing would prevail. Externally, BGP tables and the existing IP nomenclature could be kept. Such an approach would already benefit from the lightweight geometric coordinates for forwarding, but would still require $O(n)$ memory.

The best integration would thus be to modify BGP so as to fully take advantage of PIE. While this is beyond the scope of this paper, we contend that it is possible to achieve. As a simple example, consider four ASs, AS1 through AS4, with AS1 and AS4 both being connected through both AS2 and AS3. Assume that BGP is configured so as to prevent AS3 from being used as a transit AS between AS1 and AS4. Assume further that, rather than using STP to create and propagate the coordinates, we now use a BGP-like mechanism.

When AS3 receives the eBGP message from AS1 to create routing coordinates, it propagates it internally, but not through its eBGP connection to AS4. On the other hand, AS2 does according to its BGP policy. Thus, traffic from AS1 to AS4 will see AS2 as in between them in the metric space, and AS3 as in a wrong direction and routing will naturally go through AS2. The weights between ASs can be built upon the BGP attributes as well. The fact that PIE adapts well to arbitrary link costs provides good support to use it for traffic engineering.

This basic example shows that there is enough expressiveness in creating the coordinates of PIE to satisfy some basic policy mechanisms.

VII. Conclusion

We have presented and evaluated PIE, a distributed protocol that produces a greedy embedding. It does so by isometrically embedding trees in non-Euclidean spaces of dimension $O(\log^2(n))$. Each node in the graph belongs to $O(\log(n))$ trees. The greediness of the embedding allows the forwarding procedure to take any available shortcut off the trees while avoiding loops and guaranteeing the success of routing. PIE typically relaxes the *deterministic* guarantees provided by classic compact routing schemes, in order to be written as a distributed protocol. The bottomline of the good features of PIE is that these guarantees are now *probabilistic*, satisfied *with probability one* on the relevant categories of large graphs.

We have proved that PIE achieves a success ratio of 100% on any graph, that it provides polylogarithmic scalability, and we have given a logarithmic upper bound on the path stretch. We have used large-scale simulation on synthetic and real-world topologies to observe that the stretch is independent of n and that it remains extremely low, typically lower than for centralized or less scalable state-of-the-art algorithms. In addition, we have proposed an optional source-aided routing mechanism that provides significant stretch improvement with no scalability penalty.

PIE comes in a clean-slate perspective. We briefly discussed the challenges related to any replacement of the existing protocols and gave indications that such a geometric scheme could be used with traffic engineering and policy routing, making this a direction worth exploring for future work.

We can draw a few orthogonal considerations from the good stretch performance obtained by PIE. It is a direct indicator of the self-similar tree-like structure of the Internet, and it shows that the embedding has low distortion. It would thus probably suit well distance estimation tasks in the Internet, as it is required by many overlay and peer-to-peer applications.

References

[1] J. Herzen, C. Westphal, and P. Thiran, "Scalable routing easy as pie: A practical isometric embedding protocol," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, 2011, pp. 49–58.

[2] D. Meyer, L. Zhang, and K. Fall, "Report from the iab workshop on routing and addressing," http://tools.ietf.org/html/draft-iab-raws-report-01.html, Feb. 2007.

[3] C. Gavoille and M. Gengler, "Space-efficiency for routing schemes of stretch factor three," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 61–679, 1997.

[4] S. Milgram, "The small world problem," *Psychology Today*, vol. 2, pp. 60–67, 1967.

[5] R. Pastor-Satorras and A. Vespignani, *Evolution and Structure of the Internet: A Statistical Physics Approach*. New York, NY, USA: Cambridge University Press, 2004.

[6] C. Papadimitriou and D. Ratajczak, "On a conjecture related to geometric routing," *Theoretical Computer Science*, vol. 244, no. 1, 2005.

[7] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proceedings of ACM MobiCom*, 2003, pp. 96–108.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 15–26, 2004.

[9] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: of theory and practice," in *PODC '03*, 2003, pp. 63–72.

[10] S. Durocher, D. Kirkpatrick, and L. Naranayan, "On routing with guaranteed delivery in three-dimensional ad hoc wireless newtorks," in *Proceedings of ICDNC*, 2008, pp. 546–557.

[11] S. Subramanian, S. Shakkottai, and P. Gupta, "On optimal geographic routing in wireless networks with holes and non-uniform traffic," in *Proceedings of Infocom*, 2008.

[12] P. Indyk and J. Matousek, "Low-distortion embeddings of finite metric spaces," in *in Handbook of Discrete and Computational Geometry*. CRC Press, 2004, pp. 177–196.

[13] P. Maymounkov, "Greedy embeddings, trees and Euclidian vs. Lobachevsky geometry," Technical Report, available at http://pdos.csail.mit.edu/ petar/pubs.html, 2006.

[14] A. Moitra and T. Leighton, "Some results on greedy embeddings in metric spaces," *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 0, pp. 337–346, 2008.

[15] R. Kleinberg, "Geographic routing using hyperbolic space," in *Proceedings of Infocom*, 2007.

[16] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *Proceedings of Infocom 2009*, April 2009.

[17] M. B. Fragkiskos Papadopoulos, Dmitri Krioukov and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *Proceedings of Infocom*, 2010.

[18] M. Boguñá, F. Papadopoulos, and D. Krioukov, "Sustaining the Internet with hyperbolic mapping," *Nature Communications*, vol. 1, no. 6, pp. 1–8, September 2010.

[19] C. Westphal and G. Pei, "Scalable routing via greedy embedding," in *Proceedings of Infocom Mini-Conference*, April 2009.

[20] A. Gupta, A. Kumar, and R. Rastogi, "Traveling with a pez dispenser (or, routing issues in mpls)," *SIAM J. Comput.*, vol. 34, no. 2, 2005.

[21] R. Flury, S. Pemmaraju, and R. Wattenhofer, "Greedy routing with bounded stretch," in *Proc. of Infocom*, April 2009.

[22] D. Krioukov, kc claffy, K. Fall, and A. Brady, "On compact routing for the Internet," *SIGCOMM Comp. Comm. Rev.*, vol. 37, no. 3, 2007.

[23] M. Thorup and U. Zwick, "Compact routing schemes," in *ACM Symposium on Parallel Algorithms and Architectures*, 2001, pp. 1–10.

[24] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith, "S4: Small state and small stretch routing protocol for large wireless sensor networks," in *Proceedings of the 4th USENIX NSDI 2007*, April 2007.

[25] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy, "Scalable routing on flat names," in *Proceedings of Co-NEXT*, 2010.

[26] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang, "Compact routing in power-law graphs," in *Proceedings of DISC'09*, 2009, pp. 379–391.

[27] A. Brady and L. Cowen, "Compact routing on power-law graphs with additive stretch," in *ALENEX*, 2006.

[28] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, and A. Rowstron, "Virtual Ring Routing: Network routing inspired by DHTs," in *Proc. of ACM SIGCOMM'06*, 2006, pp. 351–362.

[29] M. Ghaffari, B. Hariri, and S. Shirmohammadi, "On the necessity of using Delaunay triangulation substrate in greedy routing based networks," *IEEE Communications Letters*, vol. 14, no. 3, pp. 266–268, March 2010.

[30] S. Lee, Z.-L. Zhang, S. Sahu, and D. Saha, "On suitability of Euclidean embedding for host-based network coordinate systems," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 1, pp. 27 –40, 2010.

[31] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applications," *Combinatorica*, vol. 15, pp. 577–591, 1994.

[32] R. Perlman, "An algorithm for distributed computation of a spanning tree in an extended LAN," *ACM SIGCOMM Computer Communication Review*, vol. 15, no. 4, pp. 44–53, 1985.

[33] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of SIGCOMM '99*. New York, NY, USA: ACM, 1999, pp. 251–262.

[34] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k. c. claffy, and A. Vahdat, "The internet as-level topology: three data sources and one definitive metric," *SIGCOMM Comp. Comm. Rev.*, 2006.

[35] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Internet Mathematics*, vol. 1, pp. 15 879–15 882, 2002.

[36] L. Li, D. Alderson, J. C. Doyle, and W. Willinger, "Towards a theory of scale-free graphs: Definition, properties, and implications," *Internet Mathematics*, vol. 2, p. 4, 2005.

[37] F. Chung and L. Lu, "The diameter of random sparse graphs," in *Advances in Applied Math*, pp. 257–279.

[38] Y. Shavitt and E. Shir, "Dimes: let the internet measure itself," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 71–74, October 2005.

[39] T. Bu and D. F. Towsley, "On distinguishing between internet power law topology generators," in *Proc. of Infocom*, 2002.

[40] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[41] G. Bianconi and A.-L. Barabási, "Competition and multiscaling in evolving networks," *Europhysics Letters*, vol. 54, no. 4, p. 436, 2001.

[42] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," Tech. Rep., 2002.