Overlay Addressing and Routing System Based on Hyperbolic Geometry

Cyril Cassagnes, Telesphore Tiendrebeogo, David Bromberg, Damien Magoni

Overlay Addressing and Routing System Based on Hyperbolic Geometry

Cyril Cassagnes, Telesphore Tiendrebeogo, David Bromberg and Damien Magoni
University of Bordeaux – LaBRI
Bordeaux, France
{cassagne,tiendreb,bromberg,magoni}@labri.fr

***Abstract*—Local knowledge routing schemes based on virtual coordinates taken from the hyperbolic plane have attracted considerable interest in recent years. In this paper, we propose a new approach for seizing the power of the hyperbolic geometry. We aim at building a scalable and reliable system for creating and managing overlay networks over the Internet. The system is implemented as a peer-to-peer infrastructure based on the transport layer connections between the peers. Through analysis, we show the limitations of the Poincaré disk model for providing virtual coordinates. Through simulations, we assess the practicability of our proposal. Results show that peer-to-peer overlays based on hyperbolic geometry have acceptable performances while introducing scalability and flexibility in dynamic peer-to-peer overlay networks.**

I. Introduction

Internet routing is based on forwarding tables populated by blocks of IP addresses. However, the construction and maintenance of such tables requires the use of complex routing protocols that are typically not scalable in terms of memory and CPU usage. Moreover, experience shows that the IP addressing plane is insufficient due to the semantic of an IP address being both an identifier and a locator. Many efforts are currently undertaken to define new routing schemes that make use only of local information such as the addresses of the node's neighbors.

In order to test new ideas and systems, overlay networks built on top of transport layer connections (e.g., TCP or UDP) are a very convenient solution. Based on a peer-to-peer (P2P) paradigm, each overlay node (typically an IP terminal node) runs the same code and participates to the proper functioning of the overlay. Thus overlays can easily be deployed and modified, as the researcher has the full control of the nodes (as opposed to IP routers controlled only by operators).

In recent years, many new routing models emerge to solve more efficiently the problem of routing in a graph. The idea of using the information of the node location (i.e., geographic or geometric position in a space) in order to send messages has been proposed in many papers. In this paper, we propose a P2P overlay system using virtual coordinates taken from the hyperbolic plane. The forwarding of the packets inside the overlay thus requires a dedicated addressing and routing scheme. We also let the nodes connect arbitrarily to each others which leads to an overlay having a free topology.

The remainder of this paper is organized as follows. Section II gives an overview of the related previous work. Section III highlights some geometric properties of the hyperbolic plane in the Poincaré disk model and shows some limitations incurred by its use. Section IV defines the architecture of our overlay system as well as our addressing and routing algorithms. Finally, section V presents our simulation results concerning the routing and addressing schemes in both static and dynamic contexts.

II. Related work

Many existing distributed routing schemes rely on greedy algorithms [1]–[4]. The simplest routing techniques based on geographic coordinates are greedy, in the sense that nodes always forward messages to the neighbor which is closest to the destination by using the Euclidean metric [5]–[8]. However, the greediness may be wrong when there exists a node which is nearer to the destination than all of its neighbors without itself being the destination. This node is called a local minimum and packets crossing this node will fail to reach the given destination. Face routing techniques can be used for overcoming this problem but they have poor performances, that is why some researchers have even tried to predict local minima such as Liu and Wu [9].

To avoid local minima, another solution is to define an embedding. An embedding is a graph embedded in a metric space, which is a space where the notion of distance between elements is properly defined. An embedding is greedy, if and only if greedy routing is *always* successful (i.e., the triangular inequality is respected). The notion of greedy embedding of graphs was defined by Papadimitriou and Ratajczak [10] and extended by Kleinberg [11] who proved that any connected finite graph has a greedy embedding in the hyperbolic plane. Kleinberg also showed that we can easily embed a graph greedily in the hyperbolic plane by creating a spanning tree of the graph. More recently, Westphal and Pei showed in [12] a greedy embedding on a space of dimension $O(log(n))$ with route tables of polylogarithmic size at each node thus making routing scalable. Flury *et al.* proposed in [13] the first polynomial-time algorithm that embeds combinatorial unit disk graphs into $O(log_2(n))$ dimensional space, permitting greedy routing with constant stretch.

Which metric to use is a crucial choice. For instance, in [5] the metric space is the Euclidean plane, virtual coordinates are assigned using a distributed version of Tutte's *rubber band* algorithm and the embedded graph is planar, namely

it can be drawn in the plane so that the edges are continuous curves that do not intersect each other. More recently, Moitra and Leighton [14] resolved a conjecture of [10] that every 3-connected planar graph admits a greedy embedding into the Euclidean plane. Thus, to map virtual coordinates to network nodes, one has to define a subgraph and a space. Because a metric space biases the type of the subgraph, another approach is to find an adequate metric space to avoid this issue as proposed by Goodrich [15]. However, this proposition is difficult to implement in a distributed context and this aspect has not been studied in his paper.

An embedding technique suited for a distributed implementation is proposed by Kleinberg in [11]. However, the embedding requires a full knowledge of the graph topology and this topology is considered static. Recently, Cvetkovski and Crovella [16] have complemented the work of Kleinberg with the *Gravity-Pressure* algorithm to solve the local minimum issues that arise in dynamic networks subject to node and link failures. In this paper, we follow the groundbreaking work of Kleinberg [11] by modifying his method in order to apply it to large dynamic overlay networks. We use the hyperbolic plane as our metric space for selecting virtual coordinates and we propose and evaluate a scalable algorithm for dynamically assigning those coordinates to the overlay nodes.

III. Practical use of the hyperbolic geometry

In this section, we recall some facts about hyperbolic geometry. Hyperbolic geometry is similar to Euclidean geometry in many respects. It has the concepts of distances and angles, and there are many theorems common to both. The simplest hyperbolic space is the two-dimensional hyperbolic plane $\mathbb{H}^2$ of constant negative curvature -1 as opposed to the Euclidean space which is not curved. The model that we use to represent the hyperbolic plane is called the Poincaré disk model. In this model, we refer to points by using complex coordinates. We can find in the literature all the necessary information to understand the hyperbolic plane [11], [17].

A. Properties of the hyperbolic plane

Our embedding is based on a geometric property of the hyperbolic plane which allows to create distinct areas called half planes.

1) Tiling of the hyperbolic plane: in the hyperbolic plane, we can create n half spaces pair wise disjoint whatever n. This property is the base of our embedded algorithm (red line in Figure 1). While in Euclidean space an elementary property is the impossibility to create more than two half planes without having them intersect. Another important property is that we can tile the hyperbolic plane with polygons of any sizes, called p-gons. Each tessellation is represented by a notation of the form $\{p, q\}$ where each polygon has p sides with q of them at each vertex. This form is called a *schläfli symbol*. There exists a hyperbolic tessellation $\{p, q\}$ for every couple $\{p, q\}$ obeying $(p-2) \times (q-2) > 4$. In a tiling, p is the number of sides of the polygons of the *primal* (the black edges and green vertices in Figure 1) and q is the number of sides of the polygons of the *dual* (the red triangles in Figure 1).

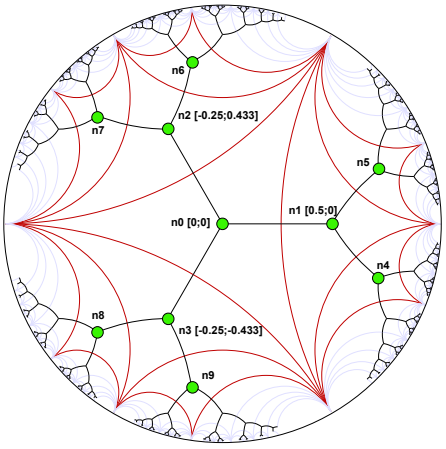

Fig. 1. 3-regular tree in the hyperbolic plane.

Our purpose is to partition the plane and address each node uniquely. That is why, we set p to infinity, thus transforming the primal into an infinite regular tree of degree q. The dual is then tessellated with an infinite number of q-gons (the red triangles in Figure 1) . This particular tiling splits the hyperbolic plane in distinct spaces and constructs our embedded tree. An example of such a hyperbolic tree with $q = 3$ is shown in Figure 1.

2) The hyperbolic distance: in the Poincaré disk model, the distances between any two points z and w are given by curves minimizing the distance between these two points and are called geodesics of the hyperbolic plane. To compute the length of a geodesic between two points z and w and thus obtain their hyperbolic distance $d_{\mathbb{H}}$, we use the Poincaré metric which is an isometric invariant:

$$d_{\mathbb{H}}(z, w) = argcosh(1 + \frac{2(|z-w|^2)}{(1-|z|^2)(1-|w|^2)}) \qquad (1)$$

The hyperbolic distance $d_{\mathbb{H}}(z, w)$ is additive along geodesics and is a Riemannian metric. For more details on the Poincaré metric we refer the reader to the proof in [17].

B. Evaluation of the practical addressing capacity

In theoretical perspective, the hyperbolic plane is unlimited. However, it is necessary to use a modeled representation of this plane and this implies the use of a precision threshold for carrying out the calculations.

1) Floating point precision issue: one property of the Poincaré model is misleading: the distances are not preserved. If we observe the Poincaré model from an outside point of view, the distance are smaller than the reality (i.e., inside the plane) because the model is a representation of the hyperbolic plane in the Euclidean plane. Indeed, the closer the points are to the boundary of the unit circle, the farther they are in reality. The hyperbolic plane has a boundary circle at infinity represented in the Poincaré disk model by a circle of radius 1

and centered on the origin $(0, 0)$. The open unit disk around the origin is the set of points whose complex modulus is less than 1.

In practice, an embedding of such a mathematical space is constrained by the precision of the floating type used, typically a *double*. This brings us to a practical concern: how can we determine the maximum number of subspaces that we can create to assign a coordinate to a node? This is a problem of arithmetic precision as we reach the maximal accuracy allowed by the calculation in floating point. The calculations usually obey to the IEEE 754 standard which determines the binary floating point representation. The floating point arithmetic can also be implemented with variable length significant numbers that are sized depending on the needs. This is called Arbitrary Precision Arithmetic (APA). As the complexity of using APA is important and as we have enough addressing capacity by using standard floating point numbers, we keep on using the classic *double* type representation. Thus two points cannot be closer that the minimum non zero *double*. Hence, the minimal half space is the space that can contain one distinct point.

To determine the maximum number of available addresses, we proceed as follows. We embed a tree with a degree of 32 and a depth equals to 32. Then, we assign an address to each node. We show in Figure 2 the gap between the number of addresses in theory and in practice. We set the maximum precision threshold to a given value and compute the addresses. We vary this maximum precision threshold from 10^{-6} to 10^{-12}. With these settings, the theoretical addressing capacity remains the same whatever the precision. We see in Figure 2 that the practical addressing capacity increases strongly between an accuracy of 6 to 9 digits. This increase is reduced a lot in the transition from 9 to 12 digits because less disjoint points can be created. Furthermore, we have observed that after 12 digits the numbers are not significant due to the rounding errors of the log and sqrt library functions. We observe a threshold where the addressing capacity is equal to 2.246E+08 with the above parameters. This upper limit is induced by the hardware and software memory restrictions of the computer doing the calculations (i.e., 4GB of RAM and about the same amount of swap file).

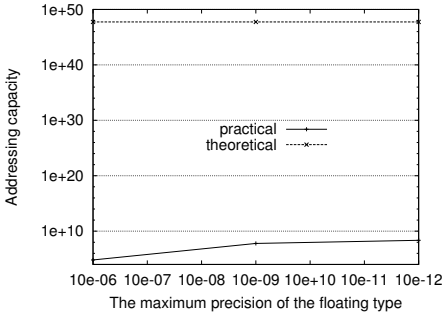

Fig. 2. Addressing capacity as a function of the floating point precision threshold.

2) Influence of the degree on the addressing capacity: The precision is set to 10^{-12} and the tree depth is set to 32 hops. The tree degree evolves from 4 to 256. Figure 3 shows that the theoretical addressing capacity increases linearly in function of the degree which is expected. We see in Figure 4 that the practical addressing capacity does not follow the same trend and remains between 2×10^8 and 3×10^8. When the degree is higher than 32 the gain in addressing capacity is weak compared to the order of magnitude observed in 3. A fine tuning of the degree parameter can improve a bit the addressing capacity as we can set the degree of the tree q to the most suitable value. This is possible because, as we create an overlay network, we have some freedom over the building of the overlay links and thus we can restrain the degree of the addressing tree. However, we can state that, whatever the q degree chosen and despite the floating point precision issues, we are guaranteed to have at least 200 million useable addresses.

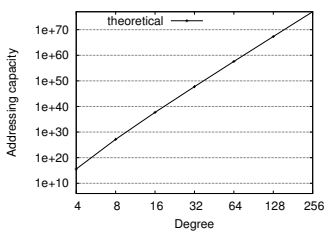

Fig. 3. Influence of the degree on the number of theoretical addresses.

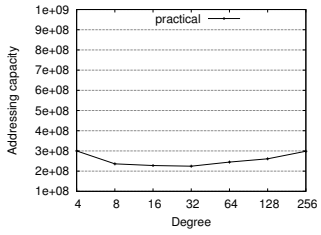

Fig. 4. Influence of the degree on the number of practical addresses.

3) Influence of the depth on the addressing capacity: To analyze the influence of the depth, we use a precision of 10^{-12} and the tree degree is set to 32. The tree depth evolves from 4 to 32. In Figure 5, we can see that the increase of the theoretical addressing capacity is exponential when the depth increases. As expected, this matches with the normal characteristics of $\mathbb{H}^2$. Because of this exponential relation between the depth and the number of addresses, in practice, the addressing capacity reaches the machine limits at 2.246E+08 with only a depth of 8.

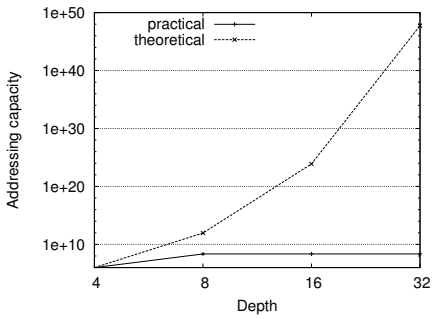

Fig. 5. Influence of the depth on the number of addresses.

IV. Addressing and routing in the hyperbolic plane

We now explain in this section how we create the hyperbolic addressing tree and how packets are routed in the overlay. We propose a distributed, dynamic and scalable routing algorithm in hyperbolic plane for Internet-wide overlay networks. We solve these two issues in this paper:

1) Each node in the graph has to compute its coordinates in the hyperbolic space without having any global knowledge of the graph topology.
2) The graph can grow and shrink over time.

A. Creating the addressing tree

We recall that the hyperbolic coordinates (i.e., a complex number) of a node of the addressing tree are used as the address of the corresponding peer in the overlay. A node of the tree can give the addresses corresponding to its children in the tree. The degree determines how many addresses each peer will be able to give. The degree of the tree is fixed at the beginning for all the lifetime of the overlay. In the overlay, a peer can connect to any other peer at any time in order to obtain an address thus fixing the degree does not prevent the overlay to grow. This specificity renders our method scalable because unlike [11], we do not have to make a two-pass algorithm over the whole network to find its highest degree.

The overlay is then built incrementally, with each new peer joining one or more existing peers. Over time, the peers will leave the overlay until there is no peer left which sets the end of the overlay. The first step in the creation of an overlay is to start the first peer and to choose the degree of the addressing tree. We nail the root of the tree at the origin of the *primal* and we begin the tiling at the origin of the disk in function of q. Each splitting of the space in order to create disjoint subspaces is ensured once the half spaces are tangent ; hence the *primal* is an infinite q-regular tree. We use the theoretical infinite q-regular tree to construct the greedy embedding of our q-regular tree. So, the regular degree of the tree is the number of sides of the polygon used to build the *dual* (see Figure 1). In other words, the space is allocated for q child peers. Each peer repeats the computation for its own half space. In half space, the space is again allocated for $q-1$ children. Each child can distribute its addresses in its half space.

The first step is thus to define the degree of the tree because it allows building the *dual*, namely the regular $q-gon$. The algorithm 1 shows how to calculate the addresses that can be given to the children of a peer. The first peer takes the hyperbolic address (0;0) and is the root of the tree. The root can assign q addresses while all the others can assign $q-1$ addresses.

This distributed algorithm ensures that the peers are contained in distinct spaces and have unique coordinates. All the steps of the presented algorithm are suitable for distributed and asynchronous computation. This algorithm allows the assignment of addresses as coordinates in dynamic topologies. As the global knowledge of the overlay is not necessary, a new peer can obtain coordinates simply by asking an existing peer to be its parent and to give it an address for itself. If the asked peer has already given all its addresses, the new peer must ask an address to another existing peer. When a new peer obtains an address, it computes the addresses (i.e., hyperbolic coordinates) of its future children. The addressing tree is thus incrementally built at the same time than the overlay.

Algorithm 1: Calculating the coordinates of a peer's children

```
CalcChildrenCoords(peer, q) ;
begin
    step ← argcosh(1/sin(π/q));
    angle ← 2π/q;
    childCoords ← peer.Coords;
    for i ← 1, q do
        ChildCoords.rotationLeft(angle);
        ChildCoords.translation(step);
        ChildCoords.rotationRight(π);
        if ChildCoords ≠ peer.ParentCoords then
            StoreChildCoords(ChildCoords);
        end if
    end for
end
```

B. Routing inside the overlay

When a new peer has connected to peers already inside the overlay and has obtained an address from one of those peers, it can start sending data packets.

Algorithm 2: Routing a packet in the overlay

```
GetNextHop(peer, packet) return Peer ;
begin
    w = packet.destinationPeerCoords;
    m = peer.Coords;
    d_min = argcosh(1 + 2 |m−w|² / ((1−|m|²)(1−|w|²)));
    p_min = peer;
    forall the neighbor ∈ peer.Neighbors do
        n = neighbor.Coords;
        d = argcosh(1 + 2 |n−w|² / ((1−|n|²)(1−|w|²)));
        if d < d_min then
            d_min = d;
            p_min = neighbor;
        end if
    end forall
    return p_min
end
```

The routing process is done in each peer on the path (starting from the sender) towards the destination by using a greedy algorithm based on the hyperbolic distances between the peers. When a packet is received by a peer, the peer calculates the distance from each of its neighbors to the destination and forwards the packet to its neighbor which is the closest to the destination as shown in the algorithm 2. If no neighbor is closer than the peer itself then the packet has reached a local minima and other methods explained in subsection IV-C must be used to successfully route the packet to the destination. If no other method is successful then the packet is dropped.

C. Coping with dynamic topologies

In a dynamic context, several problems appear. The authors of [16] say that the greediness of the embedding in [11] de-

pends critically on the connectivity provided by the underlying embedded spanning tree (or planar graph). Indeed, the routing in the hyperbolic plane is robust as long as the tree integrity is maintained. In real network environments, link and peer failures are expected to happen often. Clearly, the drawback of this method is the resilience at failures. In our overlay approach we have two levels of failures:

- The first level deals with failures in the addressing q-regular tree.
- The second level deals with failures in the overlay graph.

At the first level, if a link in the tree fails then the greedy hyperbolic routing will fail for paths taking this link. In addition, if a peer other than a leaf peer fails, this will partition the tree into a forest of up to q sub-trees and thus will disturb the connectivity of the tree [16]. We can use a recovery or maintenance technique of the tree forest with such a random walks which are a natural approach to graph exploration.

The idea is attractive but in our case we do not assure to keep the consistency addressing when the trees merge, namely not create local minimum. Indeed, the merging without up to date the nodes addresses of the sub tree is not acceptable. Furthermore, whatever the existing solution, the Achilles' heel remains the root. Moreover the random walks is not efficient in hyperbolic space because it tends to infinity. Thus to cope with failures, it is necessary to design alternative routing methods. If the addressing tree is broken, two main approaches can be used to restore the connectivity:

- Flush the addresses attributed to the nodes beyond the failed peer or link and reassign addresses to those nodes.
- Try to restore the tree by replacing the failed link by an identical new link or the failed peer by a new peer with the same connections.

The first solution that we call the *flush* method may be costly if the overlay size is very large and/or if the area beyond the failed peer/link is wide as it can lead to the renumbering of a vast part of the network. The second solution that we call the *restore* method is cheaper because the addresses are kept but it is much more difficult to implement in the case of a peer failure. Indeed, it will be hard for the new peer to set up the same connections as those of the failed peer that it replaces.

At the second level, if a link of the overlay not belonging to the addressing tree fails or if a leaf peer fails then the greedy hyperbolic routing will still work without error although the overlay paths may be longer. Figure 6 shows an examples of failure on peer $n6$. The arrows are the alternative link usable to route the packets.

The addressing tree is built on the overlay, this means that it exists underlying links, potentially shorts cut which can be became the only possible route. During the time when the above solutions are used to repair the addressing tree, other techniques can be used to ensure routing success. One technique consists, for the children losing their parent, in trying to establish overlay links to their first ancestor (i.e., grand-parent) as well as to their siblings as shown in Figure 6. If they succeed in doing this, then the hyperbolic greedy routing will be guaranteed. If they do not and the addressing tree remains broken, then the greedy hyperbolic routing may fail because of local minima issues.

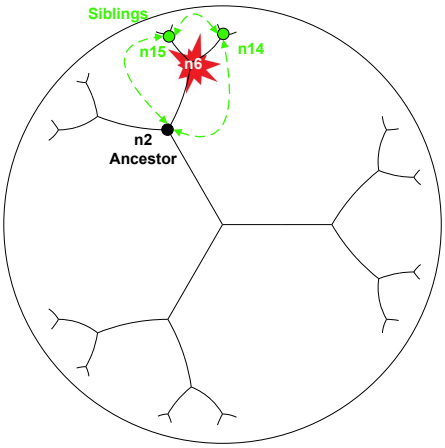

Fig. 6. Setting alternate overlay links to overcome the failure of a peer.

To overcome the problem, one such heuristic called Gravity-Pressure (GP) is presented in [16] and also used in [18]. Upon arriving in a local minimum peer, the packet enters a pressure mode. In this mode, the packet maintains a list of the nodes it has visited since it entered this mode, and the number of visits to each peer. This process continues until the packet finds a peer whose distance to the destination is smaller than the current local minimum distance. The presented solution requires the storage of variable length information in the packet header which may be difficult and cumbersome to implement.

V. Simulations

In this Section, we present the results of the simulations that we have carried out to assess the practicability, and in some cases the scalability, of our addressing and routing system based on hyperbolic coordinates. We have used a packet driven discrete event network simulator called *nem* [19] for obtaining all the results shown in this paper.

A. Settings and parameters

In order to evaluate our overlay system on realistic topologies, we have used Internet maps created from real Internet data measurements (with *nec* [20] and CAIDA [21]. We have used one IPv4 75k-node map from 2003, one BGP4 34k-node map from 2010 and one IPv6 4k-node from 2004. The floating point precision threshold is fixed to 10^{-9} for all simulations. In all simulations, the first peer creating the overlay is always a randomly picked node of the map.

In Subsection V-B concerning the static simulations, we have considered that every node of the map is a peer node that is a member of the overlay. Thus, the topology of the overlay is equal to the topology of the map and can be considered Internet-like. The simulations are defined as static because the nodes are always operational all the time and the packets

are instantly delivered between the nodes. The advantage of running static simulations is that the computation costs are low so we can use all the nodes of the map as overlay members and thus assess the scalability of our system.

In Subsection V-C concerning the dynamic simulations, we have considered that only some nodes at any given time are acting as overlay peers. The simulator's engine manages a simulation time and each overlay peer starts at a given time for a given duration on a random node of the map. The peer that creates the overlay remains active for all the duration of a simulation. The packets are delivered between the nodes by taking the transmission time of the links into account. Each simulation runs for 1 hour, thus only measurements in the middle of the simulation (around 30 minutes) can be considered as representing a steady state regime.

The number of new peers is set to 30 per minute with random inter-arrival times set with a probability following an exponential distribution. Each peer has a random lifetime set with a probability following an exponential distribution with $\lambda = 10^{-5}$ which gives a median value of 300 seconds and a 90th percentile value of 1000 seconds. As each dynamic simulation lasts for 1 hour, this distribution of the peers' session lengths produces a lot of churn. The peers create overlay links with other peers by selecting those which are closer in terms of network hops. Finally, we collect measurements every 600 seconds.

B. Static simulations of the routing schemes

In this Subsection, we use for comparison a *standard* addressing tree scheme such as the one defined in [22] and the hyperbolic addressing tree scheme previously presented in Section IV. We carry out simulations to evaluate the addressing tree depth, the stretch and the congestion metrics for both addressing schemes and on each of our three Internet maps: IPv4, BGP4 and IPv6. We study these metrics in function of the addressing tree degree.

We address all the nodes of a map by a breadth first distribution algorithm. A randomly picked first node is chosen as the root of the addressing tree. This root then computes the d coordinates of its children in the addressing tree and gives these addresses to its neighbor peers. Next, each of the d addressed children computes again $d-1$ coordinates and gives these addresses to its neighbor peers. Each child repeats the previous process until every node has an address. Each point shown on the following graphs was calculated with a confidence level of 95% and a relative statistical error of 5% (not shown on graphs).

We now study the influence of the chosen degree d on the depth of the addressing tree. The standard tree addressing scheme has variable length addresses depending on the radius of the graph. Indeed the first node is labeled 1, its fourth child will be labeled 1.4, the sixth child of this child will be labeled 1.4.6 and so on. As we use 1 byte for each level, we can see that the number of bytes in an address will depend on the maximum distance to the root. To the contrary, in the hyperbolic addressing scheme each address has a fixed length of 16 bytes (2x8-byte *double* types), however the depth of the addressing tree has a maximal value that is reached when the points are too close to the unit disk. The maximum depth observed thus depends on the degree chosen as well as the precision chosen.

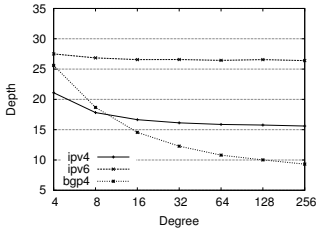

Fig. 7. Maximum depth measured when using the standard tree addressing scheme

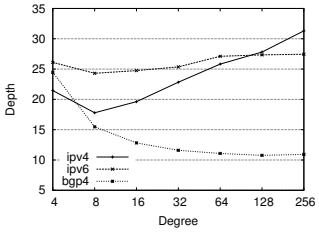

Fig. 8. Maximum depth measured when using the hyperbolic tree addressing scheme

With Figure 7 and Figure 8 we show as a function of the tree degree the standard addressing tree depth vs. the hyperbolic addressing tree depth. The degree ranges from 4 to 256. We can see that in a hyperbolic tree the results are similar to the standard tree except for the IPv4 map where the depth is greater. However this has no impact on the addresses as they have a fixed length in the hyperbolic addressing. We can conclude from these two figures that for a degree above 16, the standard tree needs addresses of length around 15-17 bytes which is close to the fixed length of the hyperbolic addresses. For IPv6 however, 27 bytes are required thus making hyperbolic addressing is a better choice.

We now study the metrics related to the routing evaluation: the stretch and the congestion. We also still show here the influence of the tree degree as an input parameter. Obviously, we do not study the success rate of the data delivery. In a static setting, this rate is always 100% as explained in Section IV. The dynamic setting will be presented in Subsection V-C.

In order to better evaluate the efficiency of these two local routing schemes we measure here the stretch of the routing paths. The stretch is equal to the local routing scheme (standard or hyperbolic) path length divided by the global routing shortest path length (i.e., the shortest possible path computed in a centralized way by the Dijkstra algorithm). The Figures 9 and 10 show that the degree has an impact on the stretch. The stretch is better when we route in the standard tree although there is a diminishing return for IP maps when the degree is increased above 16.

In Figure 10 the best stretch values for IP are respectively 1.7 with a degree of 16 and 1.5 with degree of 32. The value of the lowest stretch is 1.3 for the BGP map with a degree equal to 256. In the hyperbolic routing scheme, when the degree is higher than 32, the stretch tends to degrade and increase again for IP maps. However we must remember that the standard tree uses source routing which is less robust in presence of node failures. Furthermore, the hyperbolic greedy routing is based on a true metric (as seen in Section III) and this enables packets to take shortcuts and be more robust in presence of

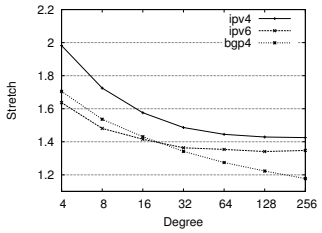

Fig. 9. Average stretch measured when using the standard source routing scheme

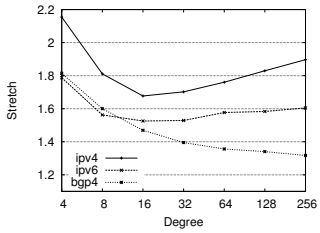

Fig. 10. Average stretch measured when using the hyperbolic greedy routing scheme

node or link failures.

As the topology of the overlay is equal to the topology of the map and that the path length between nodes is measured with our routing algorithm, we can conclude that the paths between any two nodes are farther in the hyperbolic scheme. Indeed, we remark that BGP nodes have the lowest stretch and thus the lowest path length between node which is expected as we measure AS hops and not IP hops here. The number of hops in AS maps are known to be much shorter than in IP maps. Now, we evaluate the efficiency of these local routing schemes by looking at the congestion. We define the average congestion of a node as the number of paths passing through it divided by the total number of paths. The Figures 11 and 12 show the congestion in the two cases. In both figures we observe that the congestion remains quite low. Furthermore, it is substantially the same between the two figures. However, we note that the IPv6 map has the highest congestion. This is because this map is much smaller than the other two. Finally, unlike the stretch, these plots show that the degree has a weak influence on the congestion.

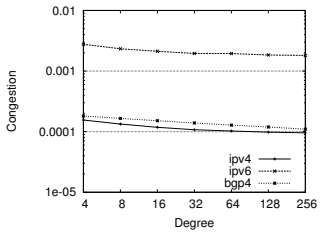

Fig. 11. Congestion measured when using the standard source routing scheme

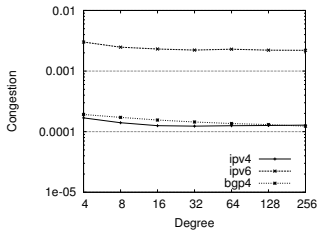

Fig. 12. Congestion measured when using the hyperbolic greedy routing scheme

C. Dynamic simulations of the routing schemes

In this Subsection, we evaluate the performance of our routing system in presence of churn. Data packets are sent by each peer (that has an address) at a rate of 1 every 10 seconds. The routing success rate for a given peer is equal to the number of data packets properly received by their destinations divided by those sent by the peer. Each point shown on the following graphs is the average value of 20 runs, and the associated standard deviation values are plotted as error bars.

In this Subsection , we only use the hyperbolic greedy routing scheme presented in Section IV. We also limit the degree to 64 because we saw in Subsection V-B that very high degrees do not improve the performances. When the addressing tree is broken because of leaving peers, the addressing tree is restored by using the *flush* method described in Section IV. None of the heuristics presented in SectionIV have been used here for improving the routing success rate. We evaluate the average routing success rate, the average path length and the stretch on the smallest IP map, that is the IPv6 map, because of the computation costs of the simulations. In Figure 13, we can see that the routing success rate is always above 90% which confirms the proper functioning of our system and the efficiency of the *flush* method in maintaining a high routing rate despite the churn.

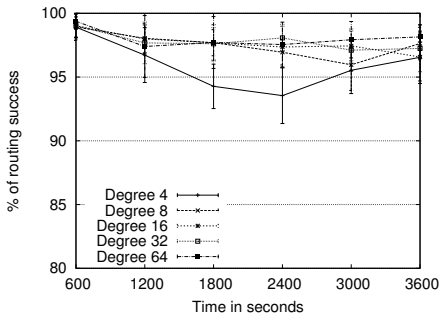

Fig. 13. Average routing success rate

Figure 14 shows the average path length of the hyperbolic routing. The path length is measured as the number of IP hops covered by the packet from the source peer to the destination peer. We can see that values are larger than the ones measured in the static simulations because here only a subset of the nodes are peers belonging to the overlay thus statistically increasing the distances. In the static simulations, the paths from all pairs were evaluated and the overlay topology was the same as the map itself. Here the nodes form an overlay which may have a different topology and thus lower path length optimality. This remains true even though overlay peers always try to establish overlay links to hop-wise closer peers.

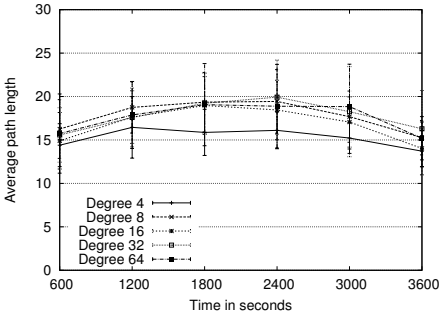

Fig. 14. Average path length between peers when using the hyperbolic greedy routing scheme

Finally, Figure 15 plots the stretch of the paths given by the hyperbolic greedy routing. As before, we define the stretch

as being equal to the hyperbolic routing path length divided by the shortest path length computed with full topology knowledge. We observe that the stretch values are much higher than in the static simulations for the same reasons as those explained above for Figure 13. For all degrees above 4, the typical stretch is around 2.6. Paradoxically for the degree 4, the stretch is lower (around 2.3) than for higher degrees which is again in opposition to what was observed in figure 10. The reason may be that when the degree increases, the peers can connect only to few other peers without having to search a lot and may end up not optimizing enough the network distances. With a low degree such as 4, peers have to ask many other peers before being able to connect to them and thus can have better peer choices with regard to the network distance. Globally in all situations the stretch remains below 3 which is an acceptable trade-off for the flexibility brought by an overlay routing system.

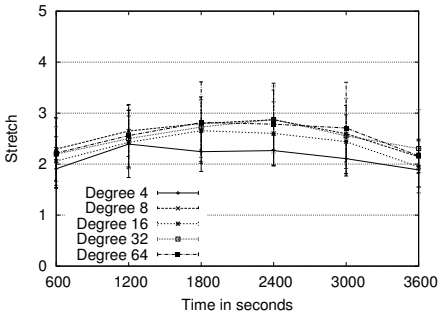

Fig. 15. Stretch measured when using the hyperbolic greedy routing scheme

VI. Conclusion

In this paper, we have proposed a dynamic P2P overlay system relying on the hyperbolic geometry. This system is able to provide addressing and routing services to all of its peers. The algorithms used inside our system are fully distributed and dynamic thus ensuring that the overlays are scalable and reliable. Our theoretical analysis of the Poincaré disk model has shown that even though we lose a huge number of potential addresses because of the floating point accuracy limitations, we can still handle a vast amount of nodes (i.e., an order of magnitude of 10^8) before reaching the limits of this model. Thus, the addressing capacity is sufficient for most overlays. Our simulation results have demonstrated in the static case that the greedy routing yields a reasonable path stretch and a low congestion ratio for overlay sizes ranging from 4k to 75k peers. They have also proved in the dynamic case that the routing success rate remains above 90% in the presence of churn. In our future work, we plan to study more complex scenarios involving data traffic between the overlay peers. Measuring the throughput between the peers is a difficult task to model and thus this will probably require the use of a real prototype. The pipelining of transport layer connections required by the overlay routing will surely affect this metric and this issue needs to be thoroughly studied.

References

[1] C. Liu and J. Wu, "Swing: Small world iterative navigation greedy routing protocol in manets," in *Proceedings of the 15th International Conference on Computer Communications and Networks*, 2006, pp. 339–350.

[2] A. Nguyen, Milosavljevic, Q. Fang, J. Gao, and Guibas, "Landmark selection and greedy landmark-descent routing for sensor networks," in *Proceedings of the 26th IEEE International Conference on Computer Communications*, 2007, pp. 661–669.

[3] S. Tao, Ananda, and C. M. Choon, "Greedy face routing with face id support in wireless networks," in *Proceedings of the 16th International Conference on Computer Communications and Networks*, 2007, pp. 625–630.

[4] A. Stavros, K. Christos, L. Ilias, and P. Evi, "An experimental study of greedy routing algorithms," in *Proceedings of the International Conference on High Performance Computing and Simulation*, 2010, pp. 150–156.

[5] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proceedings of the 9th MobiCom*. ACM, 2003, pp. 96–108.

[6] Xing, Lu, Pless, and Huang, "Impact of sensing coverage on greedy geographic routing algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 4, pp. 348–360, 2006.

[7] B. Leong, Liskov, and Morris, "Greedy virtual coordinates for geographic routing," in *Proceedings of the IEEE International Conference on Network Protocols*, 2007, pp. 71–80.

[8] Lukic, Pavkovic, Mitton, and Stojmenovic, "Greedy geographic routing algorithms in real environment," in *Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Networks*, 2009, pp. 86–93.

[9] C. Liu and J. Wu, "Destination-region-based local minimum aware geometric routing," in *Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2007.

[10] C. H. Papadimitriou and D. Ratajczak, "On a conjecture related to geometric routing," *Theor. Comput. Sci.*, vol. 344, no. 1, pp. 3–14, 2005.

[11] R. Kleinberg, "Geographic routing using hyperbolic space," in *Proceedings of the 26th IEEE International Conference on Computer Communications*. IEEE Computer and Communications Societies, 2007, pp. 1902–1909.

[12] C. Westphal and G. Pei, "Scalable routing via greedy embedding," in *Proceedings of the 28th Conference INFOCOM*, 2009, pp. 2826–2830.

[13] R. Flury, S. Pemmaraju, and R. Wattenhofer, "Greedy routing with bounded stretch," in *Proceedings of the 28th IEEE International Conference on Computer Communications*, April 2009, pp. 1737–1745.

[14] A. Moitra and T. Leighton, "Some results on greedy embeddings in metric spaces," in *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science*, 2008, pp. 337–346.

[15] M. T. Goodrich, "Succinct greedy graph drawing in the hyperbolic plane," in *Proceedings of the 16th International Symposium on Graph Drawing*, 2008.

[16] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *Proceedings of the 28th IEEE International Conference on Computer Communications*, 2009.

[17] A. F. Beardon and D. Minda, "The hyperbolic metric and geometric function theory," in *Proceedings of the International Workshop on Quasiconformal Mappings And Their Applications*, 2006.

[18] F. Papadopoulos, D. Krioukov, M. Boguñá, and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *Proceedings of the Conference INFOCOM*, 2010.

[19] D. Magoni, "Network topology analysis and internet modelling with nem," *International Journal of Computers and Applications*, vol. 27, no. 4, pp. 252–259, 2005.

[20] D. Magoni and M. Hoerdt, "Internet Core Topology Mapping and Analysis," *Computer Communications*, vol. 28, no. 5, pp. 494–506, March 2005.

[21] *The Cooperative Association for Internet Data Analysis*, University of California, San Diego, http://www.caida.org/home.

[22] K. Shami, D. Magoni, and P. Lorenz, "Autonomous, scalable, and resilient overlay infrastructure," *Journal of Communications and Networks*, vol. 8, no. 4, pp. 378–390, December 2006.