Dynamic Greedy Routing in Overlay Networks Using Virtual Coordinates from the Hyperbolic Plane

Damien Magoni, Cyril Cassagnes

Dynamic Greedy Routing in Overlay Networks Using Virtual Coordinates from the Hyperbolic Plane

Damien Magoni*, Cyril Cassagnes†

Abstract

Greedy routing algorithms based on virtual coordinates have attracted considerable interest in recent years. Those based on coordinates taken from the hyperbolic plane have interesting theoretical scalability properties. However, their scalability and reliability are yet to be ensured when applied to large scale dynamic networks. In this paper, we propose a scalable and reliable solution for creating and managing dynamic overlay networks where nodes have hyperbolic coordinates. In this context, our solution provides a greedy routing algorithm based on the hyperbolic distance. To cope with network dynamics, we have defined two methods for avoiding temporary local minima and one method for maintaining the greedy embedding over time. Through analysis, we evaluate the complexity costs of our solution. Through simulations, we assess the scalability of our solution on static networks and its reliability on dynamic networks. Results show that using our solution based on hyperbolic geometry provides scalability and reliability to both addressing and routing tasks in dynamic overlay networks.

1 Introduction

Overlay networks built on top of transport layer protocols are very useful for providing advanced services to applications. Such services include application layer multicast, multimedia streaming delivery and swarm content distribution. When an overlay is created and managed by end users as a peer-to-peer (P2P) network, it enables the sharing of the users' resources without relying on centralized corporate infrastructures such as data centers and network operators. Overlays can thus be used to extend the freedom and the flexibility of network communications. However, P2P overlay protocols often enforce specific topologies upon their participating members in order to be able to use simple routing algorithms inside their overlays. Indeed, regular routing protocols are not suitable inside overlays because of their size and churn issues. Therefore, an interesting but challenging task is to be able to build a P2P overlay which could have any topology while keeping a scalable and reliable routing solution.

For scalability purposes, *greedy routing* algorithms have the interesting property of not using any routing tables. Those algorithms only require each node to know its neighbors' addresses. Greedy routing algorithms rely on distance metrics, thus they need position-related addresses. Geographic coordinates are the simplest choice but greedy routing does not work on general topologies when geographic coordinates are used.

*University of Bordeaux, LaBRI, 351, Cours de la Libération, 33405 Talence Cedex, France. Phone: +33 540 003 540. Fax: +33 540 006 669. E-mail: magoni@labri.fr

†University of Luxembourg, SnT, Luxembourg

This is the reason why virtual coordinates are more appropriate. Indeed, they do not require any geographic knowledge concerning the nodes. In this context, the *hyperbolic plane* is useful to pick virtual addresses because it provides a proper distance metric for a greedy routing algorithm as well as a tree-like exponential addressing capability. Furthermore, a P2P overlay also has to smoothly cope with network dynamics in order to be reliable. Network dynamics consist in the setup and tear down of overlay links as well as the joins and leaves of overlay nodes at any time (also called *churn*). Because virtual coordinates depend on the relative position of a node to each others, the churn implies timely changes of the nodes' addresses inside the overlay.

Our aim is thus to provide an efficient solution for building and maintaining scalable and reliable P2P overlay networks. Our solution is based on hyperbolic greedy routing for scalability as well as distributed addressing tree maintenance and recovery for reliability. Our contributions are as follows:

- We relate our work to previous papers that provided insightful ideas as foundations for defining our solution (Section 2).
- We present some geometric properties of the Poincaré disk model of the hyperbolic plane and we define how we compute virtual coordinates (Section 3).
- We define our distributed addressing and greedy routing solution for building reliable and scalable dynamic P2P overlay networks (Section 4) including recovery methods (Subsection 4.3.1) and local minima avoidance heuristics (Subsection 4.3.2).
- We evaluate the performances of our solution over both static and dynamic overlays in order to demonstrate both its scalability and reliability (Section 5).

This paper is based on our previous work published in [1]. We have added here the definition and evaluation of two routing heuristics for circumventing local minima appearing in dynamic topologies (Subsection 4.3.2). We have also added an analysis of the complexity costs of our solution (Subsection 4.4). We have carried out new extended simulations covering real as well as synthetic topologies, and have added results concerning four routing heuristics used on dynamic overlays (Subsection 5.3).

2 Related work

Many existing distributed routing schemes rely on greedy algorithms [2, 3, 4, 5]. The simplest routing schemes based on geographic coordinates are greedy algorithms, in the sense that nodes always forward messages to the neighbor which is closest to the destination by using the Euclidean metric [6, 7, 8, 9]. 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 [10].

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. The notion of the greedy embedding of graphs was defined in 2005 by Papadimitriou and Ratajczak. They mainly investigated planar 3-connected graphs embedded in the Euclidean space [11].

Two of the earliest researchers to investigate the hyperbolic space to embed Internet maps were Shavitt and Tankel in 2004 with their work on the curvature of the Internet and its use for overlay construction and distance estimation [12, 13]. They defined the Internet geometric curvature, and showed how embedding the Internet metric in a hyperbolic space with a proper curvature gives accurate distances. Their work was later followed by Kleinberg in 2007 who proved that any connected finite graph has a greedy embedding in the hyperbolic plane [14]. Kleinberg also showed that we can easily embed a graph greedily in the hyperbolic plane by creating a spanning tree of the graph in a distributed manner. Because Kleinberg's algorithm needs to know the value of the highest degree node and does not cope with dynamic topologies, it can only be applied to small size static networks. In 2009, Cvetkovski and Crovella [15] 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.

Also in 2009, Westphal and Pei constructed in [16] 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 [17] the first polynomial-time algorithm that embeds combinatorial unit disk graphs into $O(log_2(n))$ dimensional space, permitting greedy routing with constant stretch. Other work on greedy embedding in the hyperbolic space followed in 2010 such those by Papadopoulos *et al.* who studied their use in dynamic scale-free networks [18] and by Zeng *et al.* who investigated their use for resilient routing in sensor networks [19]. An interesting idea in wireless sensor networks would be to use infection spreading [20] by following the hyperbolic addressing tree thus limiting redundant information forwarding.

Which metric to use is a crucial choice. For instance, in [6] 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 [21] resolved a conjecture of [11] 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 [22]. However, this proposition is difficult to implement in a distributed context and this aspect has not been studied in his paper.

In this paper, we follow the ground-breaking work of Kleinberg [14] 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 define a distributed algorithm for dynamically assigning those coordinates to the overlay nodes. We do not need to know the highest degree node as we fix the addressing tree degree once and for all. We can manage network dynamics by a recovery technique on the addressing tree such as our *flush* method defined below. We can overcome temporary local minima by heuristics such as *Tree Ancestor* and *Push Back* which are less costly than Gravity Pressure.

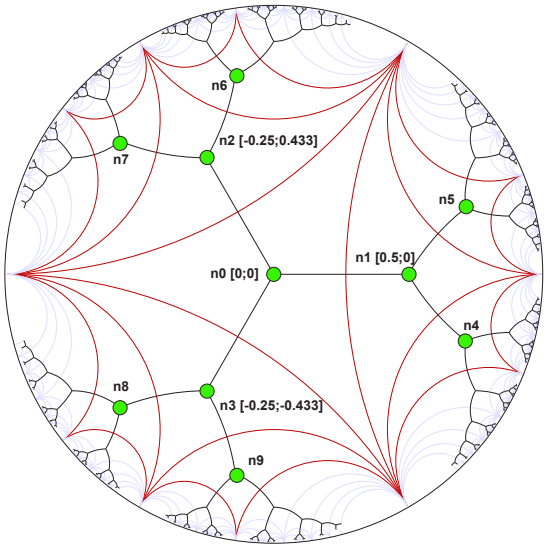

Figure 1: 3-regular tree in the hyperbolic plane.

3 Hyperbolic geometry

In this section, we recall some properties of the hyperbolic geometry and explain how we can leverage them.

3.1 Properties of the hyperbolic plane

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 H^2 of constant negative curvature -1 as opposed to the Euclidean space which is not curved. We use this hyperbolic plane as our addressing space. The model that we use to represent the hyperbolic plane is called the Poincaré disk model. In this model, the hyperbolic plane is represented by an open unit disk where the circle of radius 1 represents the infinity. Lines are arcs of circles intersecting the disk and meeting its boundaries at right angles. We refer to points of the hyperbolic plane by using complex coordinates.

3.2 Tiling of the hyperbolic plane

An important property of the hyperbolic plane is that it can be tiled 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* and q is the number of sides of the polygons of the *dual*.

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 into an infinite number of distinct spaces and constructs an infinite q-regular tree. An example of such a tree with $q = 3$ is shown in Figure 1.

We aim at creating and maintaining a large and dynamic overlay network. In order for our solution to be scalable, we define a distributed addressing scheme where a new node selects a unique address from a pool of addresses managed by an already addressed

node. The newly addressed node then derives its own pool of addresses to distribute, which is built from its own address. Therefore when the overlay grows, the addressed nodes form a spanning tree where each node address is derived from the address of its ancestor in the tree. We call *hyperbolic addressing tree* this spanning tree of addressed nodes. The hyperbolic plane enable us to easily define such pools of non overlapping addresses by mapping the addressing tree to the previously defined q-regular tree. In this context, each node in the overlay has a unique virtual address represented by a point in the hyperbolic plane and is able to distribute $q-1$ addresses to other new nodes.

3.3 The hyperbolic distance

In the Poincaré disk model, the distance between two points a and b is given by a curve called geodesic which minimizes the distance between these two points. To compute the length of a geodesic between two points a and b and thus obtain their hyperbolic distance d_H, we use the Poincaré metric which is an isometric invariant:

$$d_H(a,b) = \text{argcosh}(1 + \frac{2(|a-b|^2)}{(1-|a|^2)(1-|b|^2)}) \quad (1)$$

The hyperbolic distance $d_H(a,b)$ is additive along geodesics and is a Riemannian metric. For more details on the Poincaré metric, we refer the reader to the proof in [23].

3.4 Isometries and generators

An isometry is a distance-preserving map between metric spaces. In our context, isometries are used to determine the coordinates of the points belonging to the addressing tree. An isometry is a geometric transformation defined in the complex plane and composed of two operations: a rotation and a translation. Each operation is defined by a complex number. The first complex number r defines the rotation while the second complex number t defines the translation. Given any complex number z and an isometry $I = \{r, t\}$, then:

$$I(z) = \frac{r \times z + t}{1 + \overline{t} \times r \times z} \quad (2)$$

Given two isometries $I_1 = \{r_1, t_1\}$ and $I_2 = \{r_2, t_2\}$, then:

$$I_1 \times I_2 = \left\{ \frac{r_1 \times r_2 + r_2 \times t_1 \times \overline{t_2}}{r_1 \times t_2 \times \overline{t_1} + 1}, \frac{r_1 \times t_2 + t_1}{r_1 \times t_2 \times \overline{t_1} + 1} \right\} \quad (3)$$

Given an isometry $I = \{r, t\}$, then its inverse is defined by:

$$inv(I) = \{\overline{r}, 0\} \times \{1, -t\} \quad (4)$$

Generators are specific isometries that are used to create the points of the addressing tree. There are exactly q generators, with q being the degree of the regular tree defined above. Each generator is constructed from two isometries, a rotation isometry noted R defined as:

$$R = \left\{ e^{i\frac{2\pi}{q}}, 0 \right\} \quad (5)$$

and a translation isometry noted T defined as:

$$T = \left\{ 1, \tanh\left(\text{argcosh}\left(\frac{1}{\sin\left(\frac{\pi}{q}\right)} \right) \right) \right\} \times \{-1, 0\} \quad (6)$$

When the degree q of the addressing tree is chosen at the creation of the overlay, any node can then use Algorithm 1 to calculate the q generators.

Algorithm 1: Defining the generators.

```
DefineGenerators(Generator[], q)
begin
    R = { e^{i 2π/q}, 0 }
    T = { 1, tanh( argcosh( 1 / sin(π/q) ) ) } × {−1, 0}
    for i ← 0 to q − 1 do
        Generator[i] = R^i × T × inv(R^i)
```

4 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 two distributed algorithms, one for address allocation and the other for greedy routing. Our solution has two properties shown later on in this section:

1. Each node in the overlay can compute its pool of attributable addresses without having any global knowledge of the topology.
2. The overlay is dynamic and thus can grow and shrink over time as nodes join, leave or fail.

4.1 Distributed building of 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 node in the overlay. A node of the tree can give the addresses corresponding to its children in the tree. Thus each node will be able to give $q-1$ addresses and the root node will be able to give q addresses. The degree q of the tree is fixed at the beginning for all the lifetime of the overlay. In the overlay, a node can connect to any other node at any time in order to obtain an address thus fixing the degree does not prevent the overlay to grow.

If a node has already given all its addresses, it sends to the requesting node a reply containing the IP addresses of its neighbors, so that the requesting node can ask for addresses from any of them. This process is repeated until the requesting node is satisfied. Optimizations to this broadcast method can be done firstly by only giving the IP addresses of nodes still having available addresses and secondly by not sending all of them.

By definition, leaf nodes always have addresses to give. This specificity renders our method scalable because unlike [14], we do not have to make a two-pass algorithm over the whole network to find its highest degree. Furthermore, this would be impossible to do on large overlay networks of thousands of nodes or more.

The overlay is built incrementally, with each new node joining one or more existing nodes (it will get only one address from one of those nodes though). Over time, the nodes will leave the overlay until there is no node left which sets the end of the overlay. The first step in the creation of an overlay is to choose the degree of the addressing tree and to start the first node. The first node of the overlay always takes the address of the root node of the addressing tree (i.e., $(0,0)$)

Each node of the overlay possesses one address which is equal to a point in the hyperbolic plane. The address is composed of the coordinates of the point in the unit disk as already explained, but also of an index and an isometry. The latter two are defined

and used in order to enable the algorithm to be fully distributed (i.e., any node must store its corresponding index and isometry to be able to compute its children's addresses without the help of another node). Algorithm 2 is used to initialize the address of the root node of the addressing tree.

Algorithm 2: Initializing the parameters of the root node.

```
InitializeRootParameters(Root)
begin
    Root.Coordinates ← (0, 0)
    Root.Index ← 0
    Root.Isometry ← {1, 0}
```

A new node can obtain an address simply by asking an existing node to be its parent and to provide it with an available address. If the asked node has already given all of its addresses, the new node must ask an address to another existing node. Finally, Algorithm 3 is used by the new node when it obtains a new address in order to compute the addresses that this node will be able to give to its children. The addressing tree is thus incrementally built at the same time than the overlay. This algorithm only needs local information and thus is distributed. The fact that the addresses are taken from the q-regular tree ensures that each node has unique coordinates.

Algorithm 3: Calculating the coordinates of the children of a node.

```
CalculateChildrenCoordinates(Node, Generator[], q)
begin
    for i ← 0 to q − 1 do
        if i = 0 and Node ≠ Root then
            continue
        Child.Index ← Node.Index + i mod q
        Child.Isometry ←
        Node.Isometry × Generator[Child.Index]
        Child.Coordinates ← Child.Isometry(0)
```

Our procedure for addressing the nodes is different from the one presented by Cvetkovski and Crovella in [15]. We set the degree q of the regular tree at the beginning of the procedure and thus a parent node will never be able to provide more than q addresses to its children, whereas in their procedure, a node can add up as many children as possible until reaching the limits of floating point precision. As we consider overlay networks, we assume that a new node can look for an alternate parent if the chosen one has no more addresses to give (plus, the new node can still connect to the first one by a redundant link if possible and desired). In our procedure, we can use all the addresses of the q-regular tree whereas by construction, their procedure implies loosing some addressing space each time they add a node to a given parent node as they recursively split the addressing space in two, and get closer to the unit circle as shown on Figure 3 of paper [15]. Those two procedures show the trade-off between maximizing the number of available addresses and giving the possibility for any given parent node to connect to a theoretically unlimited number of children.

4.2 Greedy routing in the overlay network

When a new node has connected itself to nodes already inside the overlay and has obtained an address from one of those nodes, it can start sending data packets. The routing process is done in each node on the path (starting from the sender) towards the destination by using a greedy algorithm based on the hyperbolic distances between the nodes. When a packet is received by a node, the node 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 Algorithm 4. If no neighbor is closer than the node itself then the packet has reached a local minima and other methods explained in Section 4.3 must be used to successfully route the packet to the destination. If no other method is successful then the packet is dropped.

Algorithm 4: Routing a packet in the overlay.

```
GetNextHop(Node, Packet) return Node
begin
    N_min ← Node
    u ← Node.Coordinates
    w ← Packet.DestinationNode.Coordinates
    d_min ← argcosh(1 + 2|u−w|² / ((1−|u|²)×(1−|w|²)))
    foreach Neighbor ∈ Node.Neighbors do
        v ← Neighbor.Coordinates
        d ← argcosh(1 + 2|v−w|² / ((1−|v|²)×(1−|w|²)))
        if d < d_min then
            d_min ← d
            N_min ← Neighbor
    return N_min
```

4.3 Coping with dynamic topologies

In a dynamic context, several problems can appear. Authors of [15] say that the greediness of the embedding in [14] depends critically on the connectivity provided by the underlying embedded spanning tree. Indeed, the routing in the hyperbolic plane is robust as long as the tree integrity is maintained. In real network environments, link and node failures or departures are expected to happen often. Clearly, the drawback of the greedy routing is the resilience to failures. In our overlay networking approach, we define two types of failures:

- The first type deals with failures inside the addressing tree (i.e., the q-regular spanning tree).
- The second type deals with failures inside the overlay graph (i.e., of links not belonging to the addressing tree).

4.3.1 Recovery methods

In the first type of failure, if a link belonging to the spanning tree fails then the greedy hyperbolic routing will fail for all the paths taking this link. In addition, if a node other than a leaf node fails, this will partition the tree into a forest of up to q sub-trees and thus will disturb the connectivity of the tree [15]. We thus need a recovery mechanism for restoring the addressing tree. If the addressing tree is broken, two solutions can be used to restore the connectivity:

- Flush the addresses attributed to the nodes beyond the failed node 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 node by a new node with the same connections.

The first solution that we call the *flush* method may be costly if the overlay area beyond the failed node or link is large as it can lead to the renumbering of a large part of the network. When a node looses its parent, Algorithm 5 is triggered.

Algorithm 5: Loosing the parent node.

```
OnParentFailure(Node)
begin
    Node.ParentAddress ← Nil
    Tear down parent link
    Node.Address ←
    GetAddressFromNonChildNeighbor
    T ← Timer_address_search
    while Node.Address = Nil and t < T do
        Search and link to new neighbors
        Ask and get address from new neighbors
    if Node.Address ≠ Nil then
        Forward new address to its children
    else
        Node.Address ← Nil
        Node.ChildrenAddresses ← Nil
        Forward flush message to all children
    while Node.Address = Nil do
        Search and link to new neighbors
        Ask and get address from new neighbors
```

When a node receives a *flush* message, Algorithm 6 is applied.

Algorithm 6: Receiving a *flush* message.

```
OnFlushMessageReception(Node)
begin
    Node.Address ← Nil
    Node.ParentAddress ← Nil
    Node.ChildrenAddresses ← Nil
    Forward flush message to all children
    while Node.Address = Nil do
        Search and link to new neighbors
        Ask and get address from new neighbors
```

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 node failure. Indeed, it may be hard for the new node to set up the same connections as those of the failed node that it replaces. An alternative is to directly connect the ancestor of a failed node to its descendants and to also connect themselves together. Figure 2 shows in green the links that must be created in order to circumvent the failed node $n6$ and to maintain the correctness of the greedy routing algorithm.

In order to detect node failures and to trigger a recovery method, nodes regularly send keep-alive messages to their neighbors. We currently only have defined and implemented the *flush* method. If the root node fails, the first of its living children (i.e., the child having the foremost attributable address) becomes the root and executes the *flush* method to renumber the overlay. It optimizes the renumbering by attributing to any given node the same address as its previous address if possible. The *restore* method is left for future work.

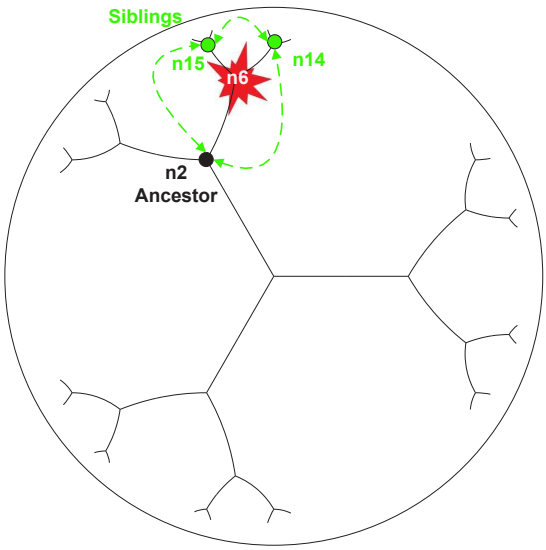

Figure 2: Setting alternate overlay links to overcome the failure of a node.

In the second type of failure, if a link of the overlay graph not belonging to the addressing tree fails or if a leaf node fails then the addressing tree will remain valid and the greedy hyperbolic routing algorithm will still work without error although the overlay paths may be longer.

4.3.2 Alternate routing heuristics

When failures of the first type occur and while the addressing tree is not restored yet (by the flush or the restore methods), the packets being routed by the greedy algorithm may get trapped in local minima. A local minimum for a packet is a node in which the packet is currently in, which is given as the best next hop node without being the destination. Local minima must be avoided in order to temporarily ensure routing success while the tree is being repaired. Any packet stuck in a local minimum is marked for alternate routing (including its current distance to destination) and is then routed by using one of these four alternate routing solutions:

- Tree Ancestor: the packet is sent to the neighbor node which is the next ancestor in the addressing tree until the reached node is closer to the destination than the packet's stored distance. If the packet reaches the root, it is deleted.
- Push Back: the packet is sent to the neighbor node whose distance to the local minimum is the biggest. When in this new node, the packet is then routed greedily as usual, unless this node is also a local minimum. In this latter case, the push back solution is applied again.
- Relaxed Greedy: the packet is sent to the neighbor node whose distance to the destination is the smallest, without looking at the current node. This solution was proposed by Krioukov *et al.* in [24].
- Gravity-Pressure: the packet maintains a list of all the nodes it has visited since it entered the alternate routing mode, as well as the count number of visits to each node. This process continues until the packet finds a node whose distance to the destination is smaller than the packet's stored distance. This solution requires the storage of variable length information in the packet header which may be difficult and cumbersome to implement. This solution was defined by Cvetkovski *et al.* in [15] and also used in [18].

As already noted in these solutions, when the packet reaches a node whose distance to destination is strictly lower than the packet's stored distance, then the packet is unmarked and proceeds with the regular greedy routing algorithm. It must be noted that the first three solutions are only heuristics and therefore they do not guarantee that the routing will succeed. The last solution is guaranteed to succeed if a path leading to the destination exists but it may store up to $O(n)$ information into the packet's header. The first three solutions may also induce loops. In order to avoid storing state information for solving loop issues, we currently use a TTL mechanism to get rid of the looping packets. The first two solutions are new and proposed by us while the last two solutions have been already proposed and defined in the literature.

4.4 Analysis of the complexity costs

We provide here a brief analysis of the complexity costs of our solution. In our solution, the nodes in the overlay connect to each others as they wish, thus no strict topology is enforced. Any node can have as many links as it wants with other nodes and one link being of course a minimum to connect to the overlay. The only requirement is that the embedded addressing tree which is a spanning tree of the overlay shall remain valid for the greedy routing to work. Because any overlay will be at least (when no redundant links exist) composed of its addressing tree, the distances between any two nodes are expected to be of the order of $O(\log(n))$ hops. If the nodes have a large number of redundant links (i.e., links not belonging to the addressing tree), the distances will be much shorter. If the overlay topology takes the form of a scale free network, the distances will be the order of $O(\log(\log(n)))$ as shown in [25]. Whatever the topology, the number of paths crossing any one node (its congestion level) will have an expected probability of at most $O(\log(n)/n)$.

Because we use a greedy routing, we do not construct and maintain routing tables and the number of states to maintain in any one node is only equal to the number of its neighbor nodes which does not grow with n thus giving a constant complexity cost being of the order of $O(1)$. When a node joins the overlay, only its neighbors (i.e., those having setup a link with the new node) need to update their state information which bears a message cost complexity independent of n. Similarly, when a node leaves the overlay, only its neighbors need to update their state information also giving a message complexity cost being of the order of $O(1)$.

However, if the addressing tree is broken and can not be restored in a reasonable amount of time, a partial readdressing of the network can occur. All nodes having addresses derived from the failed or unreachable node's address will have to be renumbered. Let's assume that the addressing tree has a degree of q and a depth of d and that each node in the network has the same probability of failing. Because of the structure of the addressing tree, the maximum probability of a failing node being at depth lower or equal to d_f (with $d_f > 0$ and $d_f < d$) is given by:

$$p = \frac{N_{d_f}}{N} = \frac{2 - q(q-1)^{d_f}}{2 - q(q-1)^{d}} \approx (q-1)^{d_f - d} \tag{7}$$

We can see that when d_f is small and d grows, the probability tends towards 0. This is expected as most nodes are peripheral in the addressing tree and their failures induce less renumbering. On the contrary, if a node close to the tree center fails (i.e., d_f small), then a large part of the tree will need to be renumbered. This latter case is expected to be uncommon given the above probability. The maximum number r of nodes to renumber will be given by:

$$r = (q-1)^{d - d_f} \tag{8}$$

If $d_f \ll d$, the message cost complexity (i.e., the renumbering procedure) is expected to be of the order of $O(q^d) \approx O(n)$.

Readdressing is needed to provide to the nodes the ability of connecting to whatever nodes they want. If we force some nodes to connect to some specific nodes for restoring the addressing tree (as done by Chord, where a node's IP address determines to which nodes it must connect) then the message cost complexity is expected to be of the order of $O(1)$ for a leaving node. Thus readdressing must be seen as a costly feature that can be opted out if performance is desired over flexibility.

4.5 Analysis of the addressing capacity

As indicated in the previous sections, the address of a node is composed of its coordinates in the hyperbolic plane. Thus an address contains two real numbers. For practical reasons, we use the usual hardware implemented `float` and `double` fundamental data types but arbitrary precision arithmetic could also be used. In order to evaluate the size of the addressing space, we have computed the number of addresses for those two cases. We have set the precision conservatively to 10^{-6} for `float` numbers and 10^{-12} for `double` numbers, to avoid the rounding errors involved by the use of mathematical functions. Addresses have a length of 64 bits with floats and 128 bits with doubles. Table 1 shows the number of addresses available given the address size. Of course, as for any addressing space, these values may not be reached if the addressing tree is not fully filled. When using floats, the floating point precision is the limiting factor: when the degree increases, the points get closer to the unit circle and their numbers is much reduced. When using doubles, this phenomenon can not be seen because the computation of the addresses has reached the memory limits of our computer (16GB of RAM). The values vary with the degree because of the way we stored the computed isometries. The values in the third column would be much higher if more memory was available and the precision would be the limiting factor. To conclude this analysis, we can see that using 128-bit addresses would provide for an addressing capacity in the order magnitude of 10^8 which should be sufficient for overlay applications.

Table 1: Addressing capacity.

Degree	`float` addresses	`double` addresses
4	999753	345064143
8	333281	271535818
16	142609	263709515
32	66049	262677051
64	32065	286537546
128	16257	311691037
256	7937	359347909

5 Evaluation

In this section, we present results obtained by simulations that we have carried out on various topologies to assess the performances and the scalability of our addressing and routing solution based on virtual hyperbolic coordinates.

5.1 Settings and parameters

In order to evaluate our overlay solution on various and realistic topologies, we have used four Internet maps created from real Internet data measurements obtained by *nec* [26] and CAIDA [27]. We have used one IPv4 75k-node map created in 2003, another 12k-node map created in 2004, one BGP4 34k-node map created in 2010 and one IPv6 4k-node from 2003. Although these maps

are not recent, they exhibit typical power laws and small world properties that have been found to be constant over time [28]. We have also generated synthetic maps by using the *nem* software [29]. More precisely, we have generated maps of sizes 5k, 10k, 20k and 40k nodes, following the Erdös-Rényi model (ER), the Waxman model (WM) and the Magoni-Pansiot model (MP). These three models present different ways of creating edges as shown in Table 2. The floating point precision threshold is set to 10^{-9} for all simulations. In all simulations, the first node creating the overlay is always randomly selected.

Table 2: Topology models for synthetic maps.

Model	Edge probability $p(u,v)$	Reference
Erdös-Rényi (ER)	C	[30]
Waxman (WM)	$\beta \exp \frac{-d(u,v)}{\alpha L}$	[31]
Magoni-Pansiot (MP)	$f(d(u,v), deg(u), deg(v))$	[32]

In Subsection 5.2 concerning the static topologies, we have considered that every node of the map is a member of the overlay. Thus, the topology of the overlay is equal to the topology of the map. The simulations are defined as *static* because the nodes are always operational all the time and the packets are instantly delivered between the nodes with no errors. 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 solution.

In Subsection 5.3 concerning the dynamic topologies, we have considered that only some nodes at any given time are acting as overlay nodes. The simulator's engine manages a simulation time and each overlay node starts at a given time for a given duration on a random node of the map. The packets are delivered between the nodes by taking the transmission time of the links into account.

5.2 Performance results of the routing schemes on static topologies

In this section, we use for comparison a *hierarchical* addressing scheme such as the one that we defined in [33] as well as the hyperbolic addressing scheme previously presented in Section 4. We carry out simulations to measure the following metrics: average path length (measured in hops), average stretch and average congestion ratio. We study these metrics in function of the degree q of the addressing tree which also corresponds to the maximum number of addresses per level (i.e., the maximum label size) in the hierarchical addressing scheme. Each point shown on the following graphs is the average value of 10^5 runs and the associated standard deviation values are plotted as error bars.

Figure 3 shows on a log-log scale the average path length observed with the hierarchical addressing. The distance is computed as the number of hops taken by the greedy routing. The smaller is the distance, the better is the routing efficiency. We see that the degree has a very small impact on the path length, reducing the path length a little when it is increased. The BGP4 map has the smallest length (around 5), which is logical as it is a map composed of Autonomous Systems where distances are small. The 40k-node Waxman map has the highest length (around 50), which is explained by the fact that links are created with a probability that decreases with the distance between the nodes. Thus, a 40k-node map exhibits long path lengths. All the other topologies have path lengths between 7 and 10, despite their sizes. This is due to the existence of shortcuts found in small world graphs as well as ER graphs (where links are randomly created thus covering long distances).

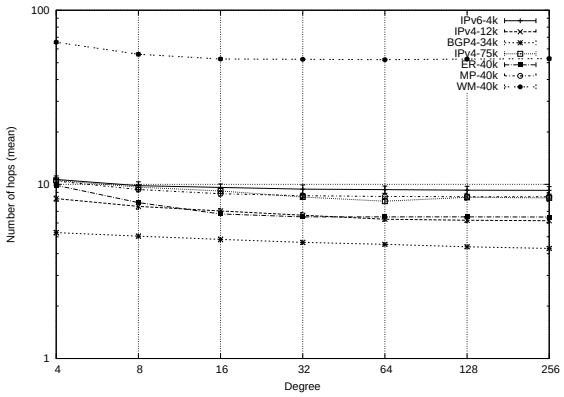

Figure 3: Average path length in hops *vs* the degree of the hierarchical addressing tree.

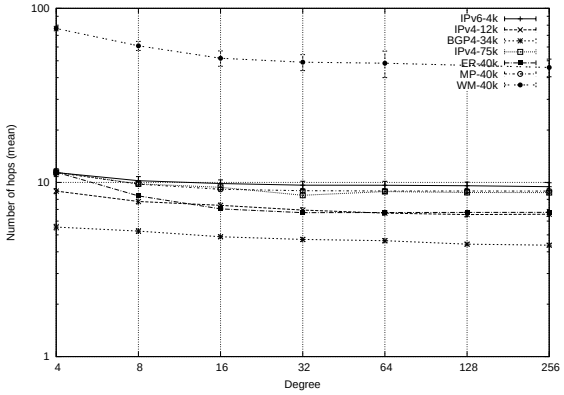

Figure 4: Average path length in hops *vs* the degree of the hyperbolic addressing tree.

Figure 4 shows on a log-log scale the average path length observed with the hyperbolic addressing. We see that the plots are very similar to the ones using the hierarchical addressing. As we have observed this phenomenon with all the other results, we only show the plots of our hyperbolic addressing system on the following figures. The 90th percentile path length values of all the plots are all below 12, except for the WM-40k map were they are below 150.

Figure 5 shows on a log-log scale the average path length observed with the hyperbolic addressing on various sizes of the synthesized maps. The MP maps show no influence of the size on the average path length. This is expected as these maps are built following an Internet-like topology and it as been shown in numerous studies that the average distance is nearly constant [26]. The ER and WM maps however, have path lengths that increase when the size increases. As the plots are linear on a log-log scale, we can deduce that the path length follows a power law with respect to the map size (i.e., $path = \alpha \times size^{\beta}$). The ER maps have a much smaller slope than the WM maps. Indeed, the latter maps have long distances between far off nodes as long links are very scarce. As ER maps have random links, they have more long distance (shortcut) links and thus a lower path inflation given the size of the map.

In order to better evaluate the efficiency of our greedy routing scheme, we measure here the stretch of the routing paths. The stretch is equal to the path length given by the greedy hyperbolic routing divided by the shortest path length given by a global routing (i.e., the shortest possible path computed in a centralized way by the Dijkstra algorithm). Figure 6 shows that the degree has an

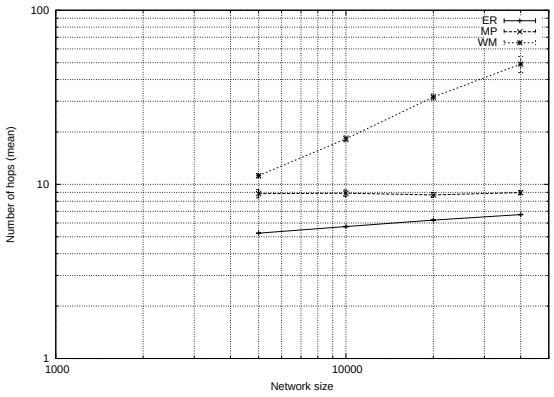

Figure 5: Average path length in hops *vs* overlay size when using the hyperbolic addressing.

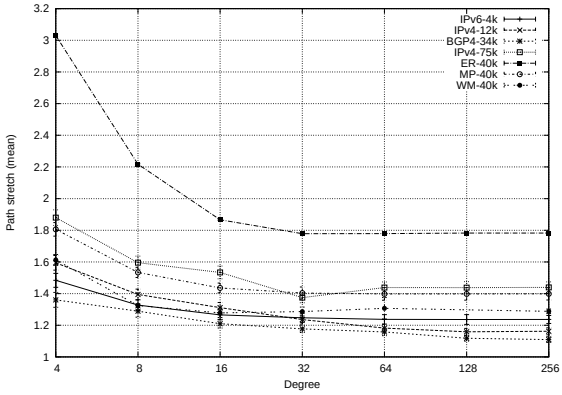

Figure 6: Average stretch measured when using the hyperbolic addressing.

impact on the stretch at low values. When the degree is between 4 to 16, the stretch gradually decreases when the degree increases. At degree 32 and above, the stretch does not vary much anymore. This can be explained by the fact that the degree is just the maximum number of addresses (minus 1) that can be attributed to the children of a node, but they may not all be given. As expected, the BGP4 map has the lowest stretch values. The ER maps on the opposite have the highest values. This means that the hyperbolic addressing tree is not able to fully leverage those topologies as they possess a lot more links compared to the other topologies. For all the other maps and for degrees above 16, we can see that the average stretch is around or below 1.4 which is good for a greedy routing on large maps. The 90th percentile stretch values for degrees above 16 of all the plots are all below 2.0, except for the ER-40k map were they are below 2.5.

We observe the average congestion ratio in the nodes. Given a set of source-destination pairs, we define the average congestion ratio of a node as the number of paths linking those pairs crossing through this node divided by the total number of paths linking those pairs in the map. Figure 7 shows the average congestion given the degree chosen. Unlike the stretch, these plots show that the degree has a very weak influence on the congestion. We notice that the IPv6 map has the highest congestion values (around 0.1), closely followed by the WM-40k map (0.06). This is due to the fact that these maps have fewer links (hence paths) with respect to their sizes. All the other maps have congestion ratio values around 0.01 which is rather low. The 90th percentile congestion ratio values of all the plots are all below 0.1, except for the IPv6 map were they are below 0.3.

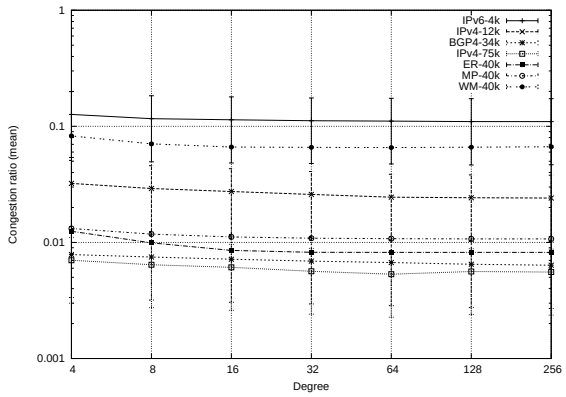

Figure 7: Congestion measured when using the hyperbolic addressing.

Results shown in this section have illustrated that our addressing and routing solution can scale for overlays having sizes up to 75k nodes. Measured path lengths between 7 to 10, stretches between 1.2 to 1.5 and congestion metrics between 0.1 to 0.01 exhibit values that are in accordance with the analysis of Section 4.4 given most topologies (except for Waxman graphs) and an implementation of our solution should therefore be able to scale to at least an order of magnitude of 10^5 nodes.

5.3 Performance results of the hyperbolic greedy routing on dynamic topologies

Results shown in this section come from simulations carried out in the OMNeT++ simulator [34, 35]. Our implementation is freely available on the Web [1].

Each simulation runs for 2 hours, thus only measurements past the beginning of the simulation (around 25 minutes or more) can be considered as representing a steady state regime. The number of new nodes trying to join the overlay each minute is dependent on their random inter-arrival times which are set with a probability following an exponential distribution with $\lambda = 0.25$. Each node has a random lifetime set with a probability following an exponential distribution with $\lambda = 2.10^{-3}$ which gives a median value of 350 seconds and a 90th percentile value of 1000 seconds. Given these values, this roughly amounts to 12% of the total of the overlay nodes joining and leaving every minute. These parameters are chosen as such because they are typical of P2P live video streaming overlays as shown in [36, 37, 38]. As each dynamic simulation lasts for 2 hours, this distribution of the session lengths produces a lot of churn. The new nodes create overlay links with other nodes by selecting those which are closer in terms of network hops. The average size of the overlay in each of these simulations is 2000 nodes. This value was the maximum we could use given our hardware as these simulations have very high computation and memory costs. We collect measurements every 100 seconds. Data packets are sent by each node at a rate of 1 every 5 seconds. The degree q of the addressing tree has been set to 5 because a low value is more realistic for a real overlay where end-nodes typically don't have much bandwidth, although the stretch will be worse as shown in the previous subsection. The routing success ratio for a given node is equal to the number of data packets properly received by their destinations divided by those sent by the node. Each point shown on the following graphs is the average value of 10 runs and the associated standard deviation values are plotted as error bars. We only use here the hyperbolic greedy routing scheme presented

[1] http://www.labri.fr/perso/magoni/sr2d3/

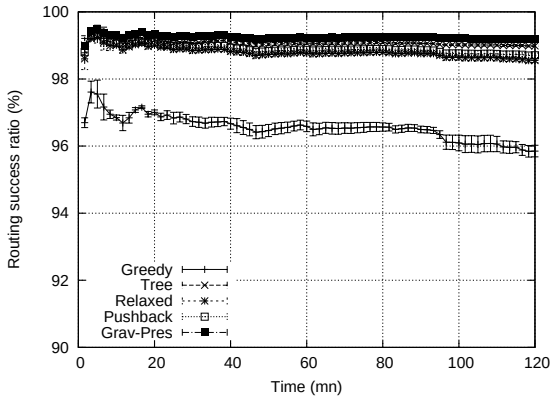

Figure 8: Routing success ratio for the alternate heuristics.

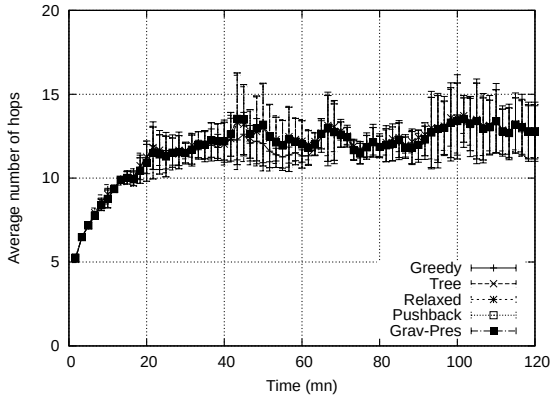

Figure 9: Average path length in hops.

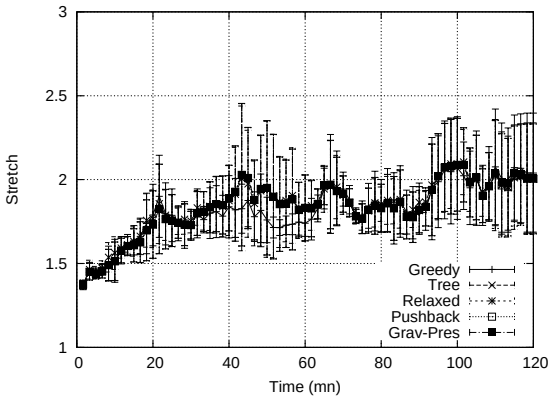

Figure 10: Average stretch.

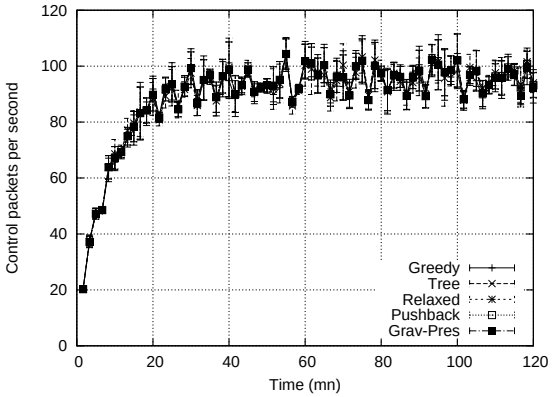

Figure 11: Number of control packets emitted per second.

in section 4. When the addressing tree is broken because of nodes leaving, the addressing tree is restored by using the *flush* method described in Section 4.

Figure 8 shows the impact of the various alternate routing heuristics on the routing success ratio. They do increase the success ratio over the greedy routing algorithm used alone. However, because the parameters have already been optimized, the gain brought by the alternate heuristics are not significant. The Gravity-Pressure method, which has been set to store at most 32 crossed nodes, gives the best results. However, the other methods nearly provide the same results. Results show that in this dynamic environment, the successful delivery of the packets can be very close to 100% when using the *flush* method and the alternate routing heuristics.

Figure 9 shows the average path length of the hyperbolic greedy routing algorithm. The path length is measured as the number of hops covered by the packet from the source node to the destination node. After the warm up phase, the path length remains roughly constant in time at around 13 hops. We can see that values are larger than the ones measured on the static maps which were around 8 to 10 hops because here only a subset of the nodes are belonging to the overlay thus this statistically increases the distances. Furthermore the churn also causes the paths to increase as local minima may have to be circumvented. They is nearly no difference between the various heuristics and the greedy routing excepted around 40 to 60 minutes where the greedy algorithm outperforms the others. This is to be expected as the alternate paths will always be longer than the greedy ones. However the success ratio will also be lower in the greedy case without heuristics.

Figure 10 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 the knowledge of the full topology. We observe that the stretch values are higher than in the static maps because the paths are increased by the alternate routing heuristics. Globally with any heuristic, the average stretch remains stable over time with values between 1.7 and 2 which is an acceptable trade-off for the flexibility brought by an overlay greedy routing solution.

Figure 11 shows the number of control packets emitted per second. Once the warm up phase is over, the amount of control traffic remains roughly constant over time at a rate of roughly 100 packets per seconds. Given that a control message has a size roughly equal to 100 bytes, this translate to a control traffic of 80kbps for a 2000-node overlay. This cost needs to be put in balance with the above 95% routing success ratio and the high churn endured by the overlay. We think that the control traffic has a reasonable cost for properly sustaining the addressing tree. We see that various heuristics have nearly no influence on the traffic. This is expected as the heuristics do not rely on control messages although they can be impacted by the duration of transitory local minima.

Results shown in this section have illustrated that our addressing and routing solution can reliably support a highly dynamic overlay averaging a size of 2000 nodes. The observed success ratio, path length, stretch and control traffic metrics remain stable over time (after a transitory start up phase) which attests for the proper maintenance of the overlay. The trade-off obtained between the success ratio, stretch and traffic control seems promising for exploitation in streaming overlay networks.

6 Conclusion

Providing advanced services to networked applications by using overlays upon the Internet is a promising way of improving network technologies. Because of its tessellation properties, the hyperbolic plane through the use of the Poincaré disk model is well suited for attributing virtual coordinates to overlay nodes thus enabling an efficient greedy embedding and routing.

In this paper, we have proposed a P2P overlay solution relying on the hyperbolic geometry for providing addressing and routing services to all the overlay nodes. The algorithms used by our solution are fully distributed and dynamic thus ensuring the scalability and reliability of the overlay.

Our analysis of the complexity costs has shown that our solution is scalable with path lengths of the order of $O(log(n))$ hops. Our simulation results have demonstrated in the case of static topologies that the greedy routing yields a reasonable path stretch and a low congestion ratio for overlay sizes ranging from 4k to 75k nodes. They have also shown in the case of dynamic topologies with 2k nodes, that the routing success rate remains above 90% in the presence of churn and that path lengths still keep their $O(log(n))$ order of magnitude.

Future work is already planned in three main directions:

- Concerning the design part, we need to investigate and design the *restore* method, that is, how to insert a new node in place of a failed node while preserving the neighboring addresses.
- Concerning the evaluation part, we need to simulate dynamic overlays of bigger sizes (at least one more order of magnitude) and with other sets of parameters (i.e., applications other than streaming) and we need to study scenarios involving more realistic data traffic models between the overlay nodes.
- Concerning a future implementation, we need to thoroughly study the pipe-lining of transport layer connections arising from the multi-hop routing in the overlay as it may affect the overall throughput.

Although those many points need further studies, we believe that the results shown in this paper are promising enough for starting an implementation.

Acknowledgment

This research work is a part of the SR2D project which is supported by the *Regional Council of Aquitaine* under grant number 20091104001.

References

[1] Cassagnes C, Tiendrebeogo T, Bromberg D, Magoni D. Overlay addressing and routing system based on hyperbolic geometry. *Proceedings of the 16th IEEE Symposium on Computers and Communications*, 2011; 294–301, doi:10.1109/ISCC.2011.5983793.

[2] Liu C, Wu J. Swing: Small world iterative navigation greedy routing protocol in manets. *Proceedings of the 15th International Conference on Computer Communications and Networks*, 2006; 339–350, doi:10.1109/ICCCN.2006.286299.

[3] Nguyen A, Milosavljevic N, Fang Q, Gao J, Guibas L. Landmark selection and greedy landmark-descent routing for sensor networks. *Proceedings of the 26th IEEE International Conference on Computer Communications*, 2007; 661–669, doi:10.1109/INFCOM.2007.83.

[4] Tao S, Ananda A, Choon CM. Greedy face routing with face id support in wireless networks. *Proceedings of the 16th International Conference on Computer Communications and Networks*, 2007; 625–630, doi:10.1109/ICCCN.2007.4317887.

[5] Stavros A, Christos K, Ilias L, Evi P. An experimental study of greedy routing algorithms. *Proceedings of the International Conference on High Performance Computing and Simulation*, 2010; 150–156, doi:10.1109/HPCS.2010.5547143.

[6] Rao A, Ratnasamy S, Papadimitriou C, Shenker S, Stoica I. Geographic routing without location information. *Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom)*, ACM, 2003; 96–108.

[7] Xing G, Lu C, Pless R, Huang Q. Impact of sensing coverage on greedy geographic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems* 2006; **17**(4):348–360, doi:10.1109/TPDS.2006.48.

[8] Leong B, Liskov B, Morris R. Greedy virtual coordinates for geographic routing. *Proceedings of the IEEE International Conference on Network Protocols*, 2007; 71–80, doi:10.1109/ICNP.2007.4375838.

[9] Lukic M, Pavkovic B, Mitton N, Stojmenovic I. Greedy geographic routing algorithms in real environment. *Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Networks*, 2009; 86–93, doi:10.1109/MSN.2009.11.

[10] Liu C, Wu J. Destination-region-based local minimum aware geometric routing. *Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2007; 1–9, doi:10.1109/MOBHOC.2007.4428646.

[11] Papadimitriou CH, Ratajczak D. On a conjecture related to geometric routing. *Theor. Comput. Sci.* 2005; **344**(1):3–14.

[12] Shavitt Y, Tankel T. On the curvature of the internet and its usage for overlay construction and distance estimation. *Proceedings of the 23rd IEEE International Conference on Computer Communications*, 2004; 384–393, doi:10.1109/INFCOM.2004.1354510.

[13] Shavitt Y, Tankel T. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Transactions on Networking* 2008; **16**(1):25–36, doi:10.1109/TNET.2007.899021.

[14] Kleinberg R. Geographic routing using hyperbolic space. *Proceedings of the 26th IEEE International Conference on Computer Communications*, 2007; 1902–1909, doi:10.1109/INFCOM.2007.221.

[15] Cvetkovski A, Crovella M. Hyperbolic embedding and routing for dynamic graphs. *Proceedings of the 28th IEEE International Conference on Computer Communications*, 2009; 1647–1655, doi:10.1109/INFCOM.2009.5062083.

[16] Westphal C, Pei G. Scalable routing via greedy embedding. *Proceedings of the 28th IEEE Conference on Computer Communications*, 2009; 2826–2830, doi:10.1109/INFCOM.2009.5062240.

[17] Flury R, Pemmaraju S, Wattenhofer R. Greedy routing with bounded stretch. *Proceedings of the 28th IEEE Conference on Computer Communications*, 2009; 1737–1745, doi:10.1109/INFCOM.2009.5062093.

[18] Papadopoulos F, Krioukov D, Boguna M, Vahdat A. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. *Proceedings of the 29th IEEE International Conference on Computer Communications*, 2010; 1–9, doi:10.1109/INFCOM.2010.5462131.

[19] Zeng W, Sarkar R, Luo F, Gu X, Gao J. Resilient routing for sensor networks using hyperbolic embedding of universal covering space. *Proceedings of the 29th IEEE International Conference on Computer Communications*, 2010; 1–9, doi:10.1109/INFCOM.2010.5461988.

[20] Bai F, Munasinghe K, Jamalipour A. Accuracy, latency, and energy cross-optimization in wireless sensor networks through infection spreading. *International Journal of Communication Systems* 2011; **24**(5):628–646, doi:10.1002/dac.1181.

[21] Moitra A, Leighton T. Some results on greedy embeddings in metric spaces. *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science*, 2008; 337–346, doi:10.1109/FOCS.2008.18.

[22] Eppstein D, Goodrich M. Succinct greedy graph drawing in the hyperbolic plane. *Graph Drawing*, *Lecture Notes in Computer Science*, vol. 5417. Springer Berlin Heidelberg, 2009; 14–25, doi:10.1007/978-3-642-00219-9_3.

[23] Beardon AF, Minda D. The hyperbolic metric and geometric function theory. *International Workshop on Quasiconformal Mappings And Their Applications*, 2006; 9–56.

[24] Krioukov D, Papadopoulos F, Boguñá M, Vahdat A. Greedy forwarding in scale-free networks embedded in hyperbolic metric spaces. *ACM Performance Evaluation Review* 2009; **37**(2):15–17.

[25] Cohen R, Havlin S. Scale-free networks are ultrasmall. *Physical Review Letters* Feb 2003; **90**(5):058 701, doi:10.1103/PhysRevLett.90.058701.

[26] Magoni D, Hoerdt M. Internet core topology mapping and analysis. *Computer Communications* 2005; **28**(5):494–506, doi:10.1016/j.comcom.2004.09.002.

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

[28] Magoni D, Pansiot JJ. Analysis of the autonomous system network topology. *ACM Computer Communication Review* 2001; **31**(3):26–37, doi:10.1145/505659.505663.

[29] Magoni D. Network topology analysis and internet modelling with nem. *International Journal of Computers and Applications* 2005; **27**(4):252–259, doi:10.2316/Journal.202.2005.4.202-1540.

[30] Erdös P, Rényi A. On random graphs i. *Publicationes Mathematicae* 1959; **6**:290–297.

[31] Waxman B. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications* 1988; **6**(9):1617–1622.

[32] Magoni D, Pansiot JJ. Internet topology modeler based on map sampling. *Proceedings of the 7th IEEE Symposium on Computers and Communications*, 2002; 1021–1027, doi:10.1109/ISCC.2002.1021797.

[33] Shami K, Magoni D, Lorenz P. Autonomous scalable and resilient overlay infrastructure. *KICS/IEEE Journal of Communications and Networks* 2006; **8**(4):378–390, doi:10.1109/JCN.2006.6182786.

[34] Pongor G. Omnet: Objective modular network testbed. *MASCOTS'93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, 1993; 323–326.

[35] Varga A. The omnet++ discrete event simulation system. *Proceedings of the European Simulation Multiconference (ESM'01)*, 2001; 1–7.

[36] Cassagnes C, Magoni D, Chang H, Wang W, Jamin S. On the Scalability of P2P-Based Push-Driven Live Streaming Systems. *Proceedings of the IEEE International Conference on Communications*, 2010, doi:10.1109/ICC.2010.5502004.

[37] Lin CS. Enhancing p2p live streaming performance by balancing description distribution and available forwarding bandwidth in p2p streaming network. *International Journal of Communication Systems* 2011; **24**(5):568–585, doi:10.1002/dac.1173.

[38] Efthymiopoulos N, Tompros S, Christakidis A, Koutsopoulos K, Denazis S. Enabling live video streaming services realization in telecommunication networks using p2p technology. *International Journal of Communication Systems* 2011; **24**(10):1354–1374, doi:10.1002/dac.1250.