Virtual Connections in P2P Overlays with DHT-Based Name to Address Resolution

Telesphore Tiendrebeogo, Daouda Ahmat, Damien Magoni, Oumarou Sié

Virtual Connections in P2P Overlays with DHT-Based Name to Address Resolution

Telesphore Tiendrebeogo
LaBRI
University of Bordeaux
Talence, France
Email: tiendreb@labri.fr

Daouda Ahmat
LaBRI
University of Bordeaux
Talence, France
Email: adaouda@labri.fr

Damien Magoni
LaBRI
University of Bordeaux
Talence, France
Email: magoni@labri.fr

Oumarou Sié
LTIC
University of Ouagadougou
Ouagadougou, Burkina Faso
Email: sie@univ-ouaga.bf

***Abstract*—Current Internet applications are still mainly bound to their transport layer connections. This prevents many features such as end-to-end security and mobility from functioning smoothly in a dynamic network. In this paper, we propose a novel architecture for decoupling communication from their supporting devices. This enforces the complete separation of devices, applications and entities such as users, services and data. Our architecture is based on a peer-to-peer overlay network where each peer has a permanent name and a variable address which depends on its position in the overlay. In order to dynamically map names to addresses, our architecture provides its own distributed hash table system. After presenting the design of our architecture, we provide a scalability analysis and by performing simulations, we assess its efficiency. Simulation results show that our overlay using a name to address resolution based on a distributed hash table is scalable and has acceptable performances given the flexibility it can provide to applications.**

Keywords-overlay; virtual connection; distributed hash table; name resolution;

I. INTRODUCTION

Current Internet communications are still based on the paradigms set by the TCP/IP protocol stack 30 years ago and they are lacking several key features. Although many efforts have been done during the last decade to provide mobility, security and multicasting, those efforts have mainly been focused on the equipment itself (e.g., computers, smartphones, routers, etc.) rather than on the logical part of the communication. In fact, although we already have a lot of mobile equipment, it is still impossible to transfer a communication from one device to another without interrupting the communication (and thus start it all over again). In the same way, although we have the choice of many applications for carrying one task, it is also still impossible to transfer a communication from one application to another without interrupting the communication. Layer 2 device mobility (e.g., WiFi, WiMAX, 3G and beyond) is nowadays well supported but users still have a very limited access to upper layers mobility (e.g., MobileIP, TCP-Migrate).

In this paper, we propose and describe a new architecture for using virtual connections setup over dynamic peer-to-peer (P2P) overlay networks built on top of the TCP/IP protocol stack of the participating devices. We have named this architecture CLOAK (Covering Layers Of Abstract Knowledge). This architecture supports names for entities (i.e., users, services, data) and devices, virtual addresses for devices, and virtual sessions for managing all kinds of Internet communications. These new semantics brought by our proposal open up many novel possibilities for such communications. The virtual connections that are setup and managed by our solution, transparently handle the break-down and restore of transport layer connections (such as TCP or SCTP connections).

This paper is an extended version of our previous work [1]. We have added here a detailed description of the Distributed Hash Table (DHT) mechanism deployed in CLOAK, an analysis of the complexity of the DHT in terms of distances, states and messages, as well as additional simulation results including comparative ones to other existing DHT systems. CLOAK was originally presented in our paper [2] which contained an extensive amount of background and related work as well as some preliminary simulation results upon static networks concerning path length. Improving upon this foundation, our paper [1] presented the protocols and modules of the architecture with greater details and reported simulation results upon dynamic networks concerning routing success ratio, path length and stretch, as well as DHT requests performance indicators. The addressing and routing system based on hyperbolic geometry which is used by CLOAK was presented in our paper [3]. Both the distributed addressing algorithm and the greedy routing algorithm are detailed in this previous paper and we have not included them here to avoid repetition. The implementation of the DHT scheme used by CLOAK over this hyperbolic system is fully explained in Section IV.

The remainder of this paper is organized as follows. Section II presents the design and features of our architecture. Section III describes the main elements of its possible implementation. Section IV presents the binding algorithm used by our DHT. Section V compares the algorithmic complexity of our proposal to those of various existing DHT systems. Section VI presents various results obtained by simulations for evaluating the routing and binding efficiency of our system. Section VII outlines the related previous work done on transport layer mobility, name and address

separation, as well as DHT schemes. We conclude the paper with a summary of our contributions and present our future research directions.

II. ARCHITECTURE

A. Design

In the context of our architecture, a *communication* is a set of interactions between several entities. It can be any form of simplex or duplex communication where information is processed and exchanged between the entities (e.g., talk, view video, check bank account, send mail, etc.). An *interaction* is simply a given type of action carried out between two or more entities by using an application protocol (e.g., FTP, HTTP, etc.). An *entity* is typically a human user but it can also be an automated service such as a server. A communication typically involves a minimum of two entities but it can involve many more in the case of multicast and broadcast communications. Finally, a device is a communication terminal equipment. On the device are running *applications* that are used by an entity to interact with other entities. Given this context, the aim of our architecture is to permit a communication to be carried out without any definitive unwanted interruption when some or all of its components (i.e., device, application or entity) are evolving (i.e., moving or changing) over space and time. Our architecture ensures that a communication has a lifetime that only depends on the will of the currently implied entities. Changes in devices, applications and even entities (when it makes sense) will not terminate the communication.

Figure 1 shows the CLOAK communication paradigm. In order to untie entities, applications and devices, CLOAK introduces the use of a *session*. A session is a communication descriptor that contains everything needed for linking entities, applications and devices together in a flexible way. A session can be viewed as a container storing the identity and the management information of a given communication. Thus the lifetime of a communication between several entities is equal to the lifetime of its corresponding session. As shown on Figure 1, a device can move or be changed for another without terminating the session. Similarly, an application can be changed for another if deemed appropriate or even moved (i.e., mobile code) also without terminating the session. Finally, entities can move or change (i.e., be transferred to another entity) without terminating the session if this is appropriate for a given communication. We can see that in our new architecture, entities, applications and devices are loosely bound together (i.e., represented by yellow arrows in Figure 1) during a communication and all the movements and changes of devices, applications and entities are supported. Note that in Figure 1, only one instance of each part (device, application, entity) of a communication is shown, other instances would obey the same scheme.

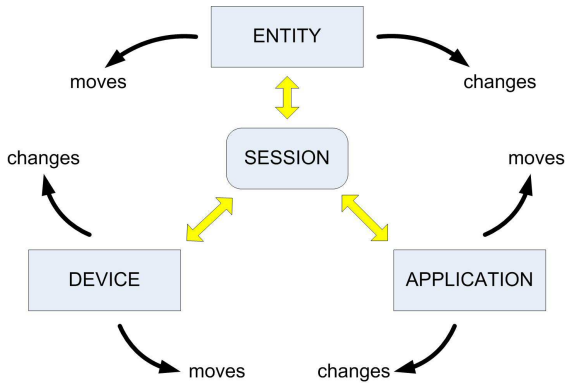

Figure 1. CLOAK communication paradigm.

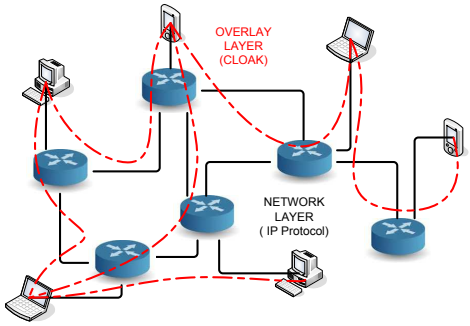

Figure 2. Overlay network.

B. Operation

In order to provide all the above mentioned features, our architecture sets up and maintains a P2P overlay network. Thus, routers are not part of the overlay. Only the devices (i.e., end-hosts or terminals) that wish to share resources in order to benefit from the architecture shall implement and run CLOAK. By doing so, they can join together to form an overlay. Figure 2 shows an overlay example with the links shown in dotted red lines. The devices connect to the others by creating virtual links (upon transport layer connections). Devices with two or more links play the role of overlay routers. The overlay network can build up without any topological constraints, as network devices can connect arbitrarily to each others and join and leave the P2P network at any time.

When joining the overlay, each device obtains a unique overlay address from one of the peers already in the overlay. The method for addressing the peers and routing the packets inside the overlay is based on the groundbreaking work of Kleinberg [4] that assigns addresses equal to coordinates adequately taken from the hyperbolic plane (represented by the Poincaré disk model). His method creates a greedy embedding upon a spanning tree of addresses (named ad-

dressing tree). This addressing tree is a regular tree of degree k. However in Kleinberg's method, the construction of the embedding requires a full knowledge of the graph topology and this topology must also be static. This is required as the degree k of the addressing tree is set to the highest degree found in the network. In our previous work [3], we have enhanced his method in order to manage a dynamic topology which is able to grow and shrink over time. Because we setup an overlay network, we are able to set the degree k of the addressing tree to an arbitrary value and as such, we are able to avoid the discovery of the highest degree node. This specificity renders our method scalable because unlike Kleinberg's method [4], we do not have to make a two-pass algorithm over the whole network to find its highest degree. The fixed degree that we choose determines how many addresses each peer will be able to give. The degree of the addressing tree is therefore set at the creation of the overlay for all its lifetime. In the overlay however, a peer can connect to any other peer at any time in order to obtain an address thus setting the degree does not prevent the overlay to grow. These hyperbolic addresses are appropriately given to the peers so that a greedy routing based on the hyperbolic distance metric is guaranteed to work when the network is connected. Thus, only the addresses of the neighbors of a peer are needed to forward a message to its destination. This is highly scalable as the peers do not need to build and maintain routing tables.

In order to set up the DHT structure needed by our architecture on top of the P2P overlay network, we only need to add a mapping function between a keyspace and the addressing space of the peers. When a peer wants to store an entry in the DHT, it first creates a fixed length key by hashing a key string with the SHA-1 algorithm. Then, the peer maps the key to an angle by a linear transformation. The peer computes a virtual point on the unit circle by using this angle. Next, the peer determines the coordinates of the closest peer to the computed virtual point. The peer then sends a store request to this closest peer. This request is routed inside the overlay by using the greedy routing algorithm presented above.

With the addressing, routing and mapping services provided by our architecture, any user/entity of the P2P overlay network can communicate with any other by setting up a virtual connection on top of the overlay. The steps for establishing a communication between two entities of an overlay are the following:

1) Bootstrap into the overlay by setting transport layer connections to one or more devices (i.e., neighbor peers).
2) Obtain an overlay address from one of those neighbor peers.
3) Identify oneself in the overlay with unique device and entity identifiers.
4) Create a session.

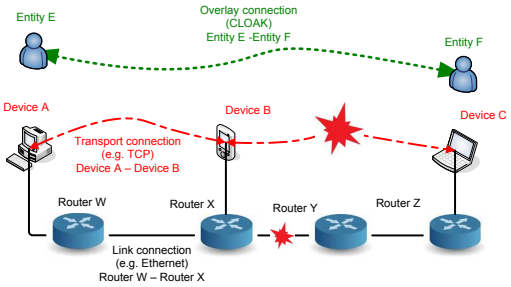

Figure 3. Virtual connections.

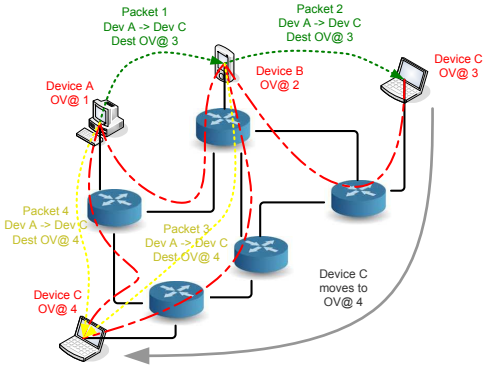

Figure 4. Steering packets inside the overlay network.

5) Contact another entity to communicate with inside this session.
6) Set an overlay layer virtual connection to this entity as shown in Figure 3.
7) Send the data stream through this connection.

If an overlay address becomes invalid, two mechanisms can be used to overcome routing failures. The first one consists, for intermediate nodes, in using the destination name inside the packet header to query the DHT for its new address. If the DHT has a more recent (and thus valid) entry, the intermediate node will then be able to update the header with the new address and forward the packet accordingly. The second one consists, for the destination node, in replacing its old address with its new one in the header of its reply packets. Upon reception, the source node will then be able to update the destination address to the newly received one. These mechanisms are illustrated in Figure 4. We call *steering*, the mechanism of querying the DHT on the fly by intermediate nodes. This mechanism also provides multicast capability when it is performed in each intermediate node. Indeed, a destination group name can be solved as several user names that again can be solved as overlay addresses.

To be able to implement our architecture, we need to introduce several new types of identifiers. More specifically we need to define the following new namespaces:

- Session namespace: any session is attributed a unique identifier that defines the session during its lifetime in the overlay.
- Device namespace: any device is attributed a unique identifier that permanently represents the device. The lifespan of this identifier is equal to the lifespan of its corresponding device.
- Entity namespace: any entity is attributed a unique identifier that represents the entity in a given context. It can be the name of a real person (John Smith) but it could also be the identifier of a professional function (Sales Manager) or the name of an organization (Michelin Company) or a specific service (Areva Accounting service). The lifespan of this identifier is equal to the lifespan of its corresponding entity.
- Application namespace: any application used during a part or all of a session is attributed a unique identifier for receiving data from the other applications of this session. The lifespan of this identifier is equal to the lifespan of the use of the application. If the entity switches to another application, this identifier is updated.

The identifiers will be stored in a DHT built on the P2P overlay network. Each peer will store a fraction of all the records in its naming module. There will be records for the devices (containing pairs like: device ID - overlay address), for the entities (containing pairs like: entity ID - device ID), for the applications (containing pairs like: application ID - session ID) and finally for the sessions (containing pairs like: session ID - session data information). An application using CLOAK will not directly open a connection with an IP address and a port number as with the usual sockets API but it will use the destination's entity ID as well as a stream ID. Figure 5 shows a typical scenario relying on this naming system for solving an entity's location. The yellow oval represents the CLOAK DHT. An entity B registers itself in the DHT by providing the device identifier it is on and its overlay address. Any entity A can now retrieve the location of B by querying the DHT. It can then connect to B via the overlay. When B switches to another device during the same session, A can reconnect to B by using its new overlay address.

As defined earlier, a session is a communication's context container storing everything necessary to bind together entities, applications and devices that are involved in a given communication. Any device, application or entity can be changed or moved without terminating the session. In order to make this possible, the session will be stored in the DHT built by the peers of the overlay network. The DHT will ensure reliability by redundantly storing the sessions on several peers. This session management system ensures the survival of the session until all the entities involved decide to stop it. Figure 6 shows a typical scenario relying on this

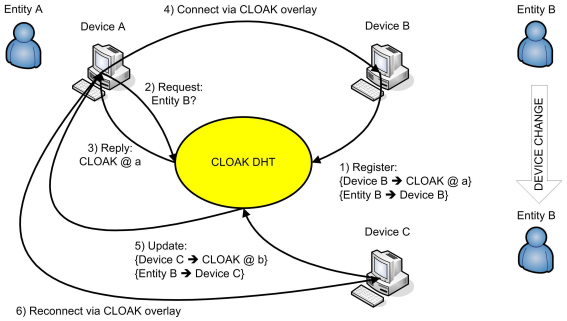

Figure 5. Identification and localization.

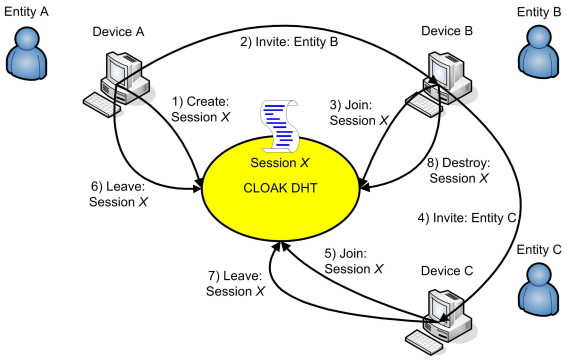

Figure 6. Session management.

session management system. The yellow oval represents the CLOAK DHT. Let us assume that an entity A wants to start a video conference communication with an entity B. It first creates a session called X describing the desired interaction (e.g., video conference) as well as the destination entity that it wants to communicate with (here the entity B). Then A sends an invite message to B that replies by joining the session X. Later on the entity B invites another entity C to participate in the video conference. C accepts and joins the session X. Three entities are now involved in the session X. Later on, the entity A leaves the session X without preventing the others to continue. This thus does not end the session X. Later on the entity C leaves the session X. The entity B being the last one involved decides to destroy the session and thus to end the communication.

C. Usage

Our architecture has a wide range of usages. It provides mechanisms for mobile and switchable applications, for adaptive transport protocol switching and for defining and using new namespaces. It can build scalable and reliable dynamic Virtual Private Networks, define fully isolated Friend-to-Friend networks, or be used as a convergence layer for IPv4, NATs and IPv6. The Table I shows the benefits of *cloaked* applications. Applications are grouped by families. Messaging applications contain e-mail, talk and chat programs. Conferencing applications regroup real-

Table I
FEATURES FOR *cloaked* APPLICATIONS.

Applications	Messaging	Conferencing	Sharing	Streaming
Reachability	✓			
Mobility		✓		✓
E2E privacy	✓	✓		
E2E auth.	✓	✓		
Pseudonymity			✓	✓
Redirection	✓			✓
Multicasting		✓	✓	✓

time audio and video communications based on signaling protocols such as SIP [5] and H.323 [6]. Sharing applications encompass file-sharing, blogging and social networking applications. Finally, streaming applications contain audio and video broadcasting services such as Internet radios, IPTV, and VoD. Most of the features are usually self-explaining but we give a few examples to highlight possible scenarios. Reachability is the ability to be reached on whatever device the user is currently using. When someone sends a message to an entity, the CLOAK DHT can be used dynamically to determine on which device is the entity and the message is routed to the proper device. Mobility is the ability of CLOAK to hide the handovers of the lower layers to the applications. If an entity is moving or switching devices, real-time applications will be maintained without interruption at the application level. CLOAK can secure connections by using entity IDs rather than device IDs (or IP addresses such as in IPsec), thus establishing End-to-End (E2E) encryption and authentication. The public keys of the peers can be stored in the DHT, however the certification of these keys must be done by a trusted third party. Because CLOAK packets usually transit through several terminals before reaching destination, the IP address of the source is often unknown to the destination thus providing partial pseudonymity. Redirection is the ability to forward a message or a stream to another entity. Finally, multicasting support is provided by CLOAK as group names can be easily set up in the DHT. This feature is useful for saving bandwidth during group communications.

III. IMPLEMENTATION

Figure 7 shows the OSI layers where the CLOAK architecture fits in. CLOAK uses the session layer and the presentation layer between the transport and application layers. These layers do not exist in the Internet stack model but they do already exist in the OSI model. In these two layers we add two new protocols. We add a CLOAK session protocol (CSP) at the session layer and a CLOAK interaction protocol (CIP) at the presentation layer. We also define new identifiers to be used by these new protocols. These new identifiers enable data streams to be bound to entities instead of network identifiers (i.e., IP address, protocol n°, port n°). As shown in Figure 1, identifiers for devices, applications and entities are interwoven together inside a session, but

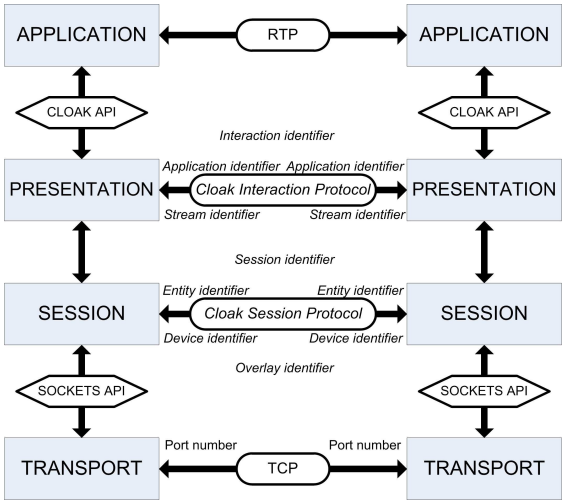

Figure 7. CLOAK architecture in the OSI model.

for the purpose of implementation, we have to order them. We chose to manage a session and its involved devices at the session layer. We also chose to manage the interactions between entities at the presentation layer. As previously said, an interaction is a type of action carried out between two or more entities. It is equal to the use of an existing application layer protocol (e.g., FTP, SMTP, HTTP, etc.). Indeed, our architecture will use the existing application layer protocols as well as the existing transport layer protocols. Thus a file transfer (FTP [7]) client application will still use the FTP protocol to speak to a FTP server. Only the portion of code for establishing a session and thus a connection to the server will have to be rewritten for using the CLOAK API instead of the socket API [8]. The code implementing the application layer protocol will not have to be changed. Please note that the CLOAK API and the mapping of application connections to transport sockets inside the middleware are not defined yet. They will be presented in a future work.

We have shown in Figure 7 how the CLOAK architecture fits in the network protocol stack. We will show how this design translates into the format of the packet headers. Figure 8 shows a CLOAK packet exchanged between a Web client and a Web server. The application header involving the HTTP protocol is now located after the CLOAK headers. We have added two additional headers. The CSP header is located directly above the TCP protocol managing the connection in the operating system of the device. It contains the overlay addresses for routing inside the overlay and enabling device mobility, the device identifiers for switching devices and enabling entity mobility and the entity identifiers for switching entities. The CIP header is located between the CSP and the application level header. It is used for identifying streams and applications. The stream identifiers are used as virtual port numbering on top of the entity. The application identifiers are used for selecting or switching

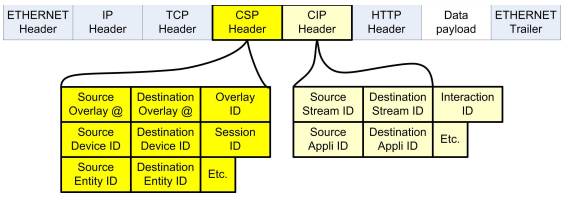

Figure 8. CLOAK protocol encapsulation.

applications when it makes sense in a communication.

The definition and implementation of the CLOAK additional protocols (CSP and CIP) and their corresponding headers will help our architecture to solve some NAT issues because applications using CLOAK will not use IP addresses and ports numbers for setting up or managing connections. They will use unique permanent entity identifiers, thus restoring the end-to-end principle of the Internet communications. The CLOAK architecture will also solve some firewall issues because any type and any number of transport layer connections can be used to connect to a CLOAK overlay. A transport layer connection can act as a multiplex tunnel for the applications using CLOAK. Thus on a given device, the applications can even use only a single port number and a single transport protocol if this is required by the firewall of the device. Indeed, a CLOAK packet has a session ID field and two stream ID fields that enable numerous applications to be multiplexed on a single transport connection if necessary. CLOAK also solves some security issues because security will be implemented by using entity identifiers instead of device identifiers or IP addresses. The security will then be independent from the devices and applications involved. Figure 9 shows the modules composing the CLOAK middleware. The functionalities provided by each module are:

- Bootstrap: primitives for creating a new or joining an existing CLOAK overlay.
- Link: primitives for managing overlay links (i.e., transport layer connections) with the neighbor peers.
- Address: primitives for obtaining an overlay address from an addressing tree parent and for distributing overlay addresses to the addressing tree children.
- Route: primitives for routing the overlay packets with the greedy algorithm using the hyperbolic distance metric.
- Steer: primitives for rerouting overlay packets by using their device or entity identifiers to update their overlay destination address.
- Connect: primitives for establishing and managing overlay virtual connections (i.e., CLOAK layer connections) to other entities.
- Bind: primitives for querying the DHT of the overlay.
- Name: primitives for managing the identifiers used by the peer.

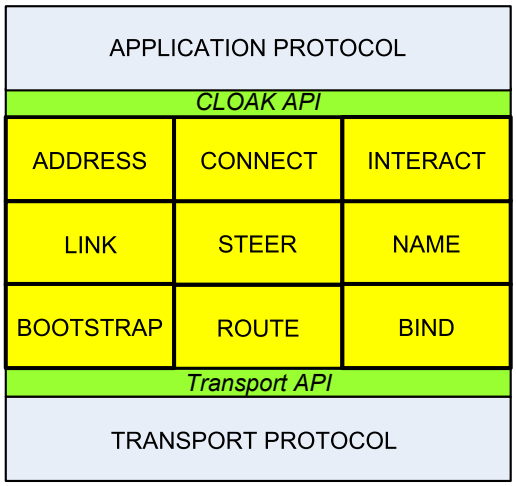

Figure 9. Modules of the middleware.

- Interact: primitives for managing the bindings between the data streams and the applications.

IV. DHT-Based Name to Address Storage

In this section we explain how our overlay system stores and retrieves the (name, address) pairs. Our solution is a structured DHT system that uses the distributed addressing and the greedy routing algorithms presented in our previous work [3].

On startup, each new member of the overlay chooses a name that identifies the device it runs on. This name will be kept by the device during all the lifetime of the overlay. When a new node obtains an address, it stores its name and its address in the DHT, with the name being used as the key and the address as the value. If the same name is already stored in the DHT, an error message is sent back to the node in order to ask the node to select another name. Thus the DHT structure itself ensures that names are unique.

A (key, value) pair is called a *binding*. Figure 10 shows how and where a given binding is stored in the overlay. A binder is any peer that stores these pairs. The depth of a peer in the addressing tree is defined as the number of parent peers to go through for reaching the root of the tree (including the root itself). When the overlay is created, a maximum depth for the potential binders is chosen. This value is defined as the *binding tree depth*. All the peers that have a depth less or equal to the *binding tree depth* in the addressing tree may hold bindings and thus be binders.

When a new peer joins the overlay by connecting to other peers, it obtains an address from one of these peers and it stores its own binding in the system. When a peer wants to store an entry in the DHT, it first creates a key by hashing the name string with the SHA-1 algorithm. It then divides the resulting 160-bit key into r equally sized $160/r$-bit subkeys (for redundancy storage). This r factor is chosen arbitrarily

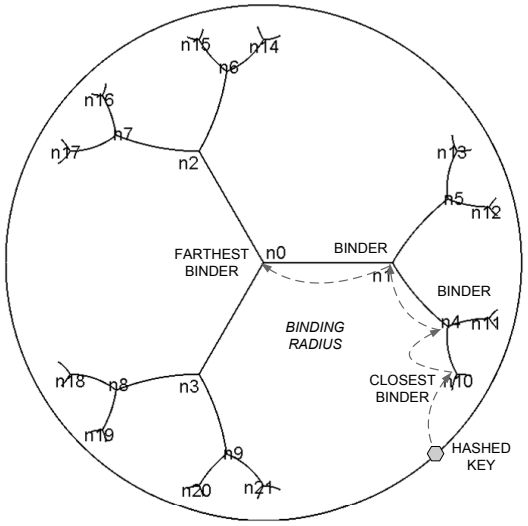

Figure 10. DHT system over the regular spanning tree.

and can be set to whatever value depending on the amount of redundancy required. In absence of redundancy, the peer selects the whole key. Then the (sub)key is mapped to an angle by a linear transformation. The angle is given by:

$$\alpha = 2\pi \times \frac{160/r\text{-bit subkey}}{\underbrace{111...11}_{r}} \quad (1)$$

The peer then computes a virtual point v on the unit circle by using this angle:

$$v(x,y) \text{ with } \begin{cases} x = cos(\alpha) \\ y = sin(\alpha) \end{cases} \quad (2)$$

Next the peer determines the coordinates of the closest binder to the computed virtual point above by using the given *binding tree depth*. In the figure we set the *binding tree depth* to three to avoid cluttering the figure. It's important to note that this closest binder may not exist in reality if no peer is currently owning this address. The peer then sends a store query to this closest peer. This query is routed inside the overlay by using the greedy algorithm presented in our previous work [3]. If the query fails because the binder does not exist or because of node/link failures, it is redirected to the next closest binder which is the father of the computed binder. This process continues until the query reaches an existing binder peer which can be any peer on the path from the computed closest binder to the center peer. Upon reaching an existing binder, the pair is stored in that binder. The query can thus go up the addressing tree to the center peer having the address (0;0) which is the farthest binder. The path from the computed closest binder to the farthest binder is defined as the *binding radius* because it is a shortest path from the edge of the disk to its center.

This process ensures that the pairs are always stored first in the binders closer to the unit circle and last in the binders closer to the disk center. If the addressing tree is imbalanced, many pairs may be stored in peers close to the center thus overloading them. In order to solve this issue any binder peer will be able to set a maximum number of stored pairs and any new pair to store will be rejected and the query redirected as above. Furthermore, to provide redundancy, the peer can repeat the storing process described above for each of the other $r-1$ subkeys. Thus r different binding radiuses can be used and this will improve the evenly distribution of the pairs. In addition, and still for redundancy purposes, a pair may be stored in more than one peer of the binding radius. A binder could store a pair and still redirect its query for storing it in its other ancestor binders. The number of stored copies of a pair along the binding radius may be an arbitrary value set at the overlay creation. We have thus defined two redundancy mechanisms for storing copies of a given binding:

1) We can use one or more binding radius(es) by creating r uniformly distributed subkeys.
2) We can store the pair in one or more binder(s) of the same binding radius.

These mechanisms enable our DHT system to cope with an non-uniform growth of the overlay and they ensure that a pair will be stored in a redundant way that will maximize the success rate of its retrieval. The number r of subkeys and the number of copies in a given radius are parameters that can be set at the creation of the overlay. Increasing them leads to a tradeoff between improved reliability and storage space cost in binders.

Our solution has the property of consistent hashing: if one peer fails, only its pairs are lost but the other binders are not impacted and the whole system remains coherent. However, this property does not hold true when a partial readdressing takes place as explained in our previous paper [3]. In this case, all the pairs stored in the peers having addresses derived from the failed or unreachable peer's address are lost. To solve this issue, as in many existing systems, pairs will be stored by following a hybrid soft and hard state strategy. Thus a pair will have to be stored by its creator every δt period of time otherwise it will be flushed by the binders that store it. These periodic store messages will ensure that pairs lost by a partial readdressing will be restored after at most δt period of time. A delete-message may be sent by the creator to remove the pair before the end of the period. We analyze the influence of the degree of the addressing tree on the query success rate and the query path length in Section VI.

V. Scalability Analysis of our DHT System

We provide in this section a brief complexity analysis of our proposal and compare the results with other existing DHT systems. We first define the four metrics that we use

in our analysis. These metrics were defined and used in the survey of Lua *et al.* [9].

1) Hops: this metric counts the average number of peers to go through to reach the destination. It is also named: path length, routing distance or dilation.
2) Paths: this metric counts the average number of paths that are crossing any given peer. It is also named congestion.
3) States: this metric counts the number of states that must be stored in a peer for the routing to work, it is typically equal to the number of entries found in the routing table of the peer. It is also named routing or memory states.
4) Churn messages: this metric counts the average number of messages that are exchanged when a peer joins or leaves the overlay. It is also named join/leave peers or linkage.

In our system, the peers in the overlay connect to each others as they wish, thus no strict topology is enforced. Any peer can have as many links as it can with other peers and one link is 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 peers 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 [10], the distances will be the order of $O(\log(\log(n)))$ as shown in [11]. Whatever the topology, the number of paths crossing any one peer (its congestion level) will have an expected probability of at most $O(\log(n)/n)$.

When a peer joins the overlay, only its neighbors (i.e., those having setup a link with the new peer) need to update their state information which bears a message cost complexity independent of n. Similarly, when a peer 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 cannot be restored in a reasonable amount of time as explained in our previous paper [3], a partial readdressing can occur for peers having addresses derived from the failed or unreachable peer's address. In this latter case, which is expected to be very uncommon, the message cost complexity is expected to be of the order of $O(n)$.

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

Table II
EXPECTED PERFORMANCE MEASURES OF VARIOUS DHT SYSTEMS.

Lookup	Hops	Paths	States	Churn messages
CAN	$O(n^{(1/d)})$	$O(n^{(1/d)}/n)$	$O(1)$	$O(1)$
Chord	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log^2(n))$
CLOAK	$O(\log(n))$	$O(\log(n)/n)$	$O(1)$	$O(1)/O(n)$
Kademlia	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$
Pastry	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$

Because we use greedy routing, we do not construct and maintain routing tables and the number of states to maintain in any one peer is only equal to the number of its neighbor peers which does not grow with n thus giving a constant complexity cost being of the order of $O(1)$.

Table II compares the complexity costs of the four above defined metrics of various DHT systems including our solution. For CAN, d is an integer equal to or greater than 2 and thus $0 < 1/d < 1$. The results presented in this table have been gathered by using the data published in [9] as well as from our previous analysis. Note that $\log$ functions with different constant bases are considered equivalent.

VI. SIMULATIONS

In this section, we present the preliminary results of the simulations that we have carried out to establish a proof-of-concept of our dynamic P2P overlay architecture. We have used our packet-driven discrete event network simulator called *nem* [12] for obtaining all the results shown in this paper.

A. Parameters

In order to evaluate our overlay system on a realistic topology, we have used a 4k-node IP level Internet map created from real data measurements with the *nec* software [13]. In all simulations, the first peer creating the overlay is always a randomly picked node of the map. We have considered that only some nodes of a map 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. Peers bootstrap by contacting the node that holds the peer that created the overlay, search for other peers to which they can connect, obtain an address from one of the peers they are connected to and send data or requests messages. This process models the birth, life and death of the overlay.

In any dynamic simulation, there is a warm up phase at the beginning and a cool down phase at the end that must both be considered as transitory regimes. Indeed, at the beginning

only the creator peer exists before new peers start and join it. Similarly, at the end, all peers are gradually leaving the overlay until only the creator peer is left and then it stops. 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. This comment must be taken into account when looking at all the plots below as most of them show a curve with a typical plateau in the middle. The most significant measurements are those located in this flat part of the plots although the other measurements are also valid.

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 $\lambda = 10e - 5$ 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. Results

We evaluate here the performances of the overlay routing depending on the chosen fixed addressing tree degree as explained in II-B. Data packets are sent by each peer at a rate of 1 every 10 seconds. We only want to evaluate routing success, query success and path lengths but not bandwidth or throughput for now that is why we do not use more realistic generated traffic patterns. 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. We observe the average routing success rate, the average path length and the 90th percentile path length as a function of the addressing tree degree of the overlay. In Figure 11, we can see that the routing success rate is always above 90% which confirms the proper functioning of our system which maintains a high routing success rate despite the churn.

Figure 12 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 presented in our previous work [3] 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

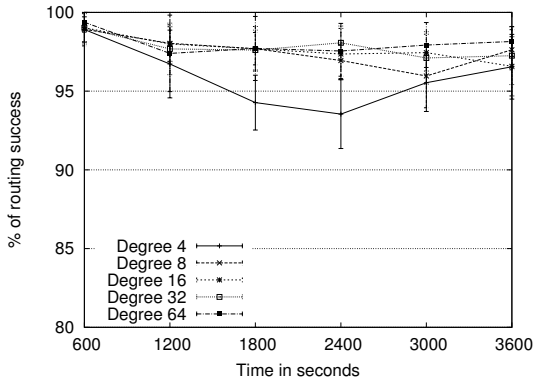

Figure 11. Average routing success rate.

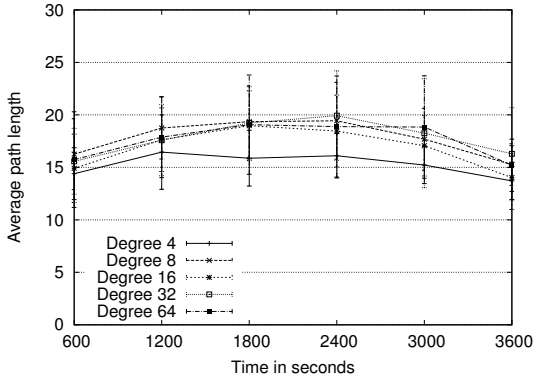

Figure 12. Average path length between peers.

peers always try to establish overlay links to hop-wise closer peers.

Figure 13 shows the 90th percentile value of the path length. Here also, the path length is measured as the number of IP hops covered by the packet. This value gives an acceptable statistical upper bound on the path length by excluding extreme cases. We can observe that the path length, for degrees above 4, is around 35 compared to the average path length of 18 seen in Figure 12. We conclude that including the values from the median to the 90th percentile yields a path inflation of 100% (i.e., paths are twice as long as the shortest ones) which is important but comparable to values measured at the IP layer [13].

We now evaluate the DHT efficiency. The only difference with the previous simulations is that now the peers do not send data packets but only storing and solving requests. The frequency of the storing requests generated in each peer is 1 every 30 seconds. The frequency of the solving requests generated in each peer is 1 every 5 seconds. We do not consider any redundancy parameters for now. Thus, a given pair is stored on one peer only. We observe the influence of the addressing tree degree of the overlay on

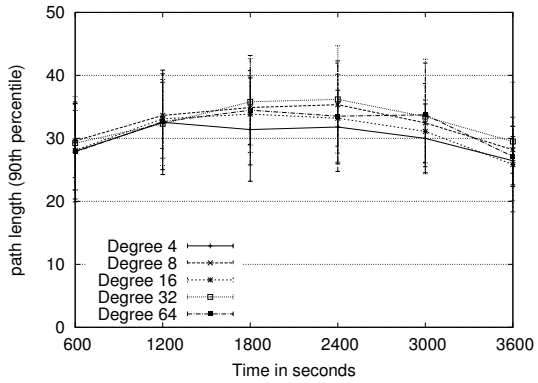

Figure 13. 90th percentile path length between peers.

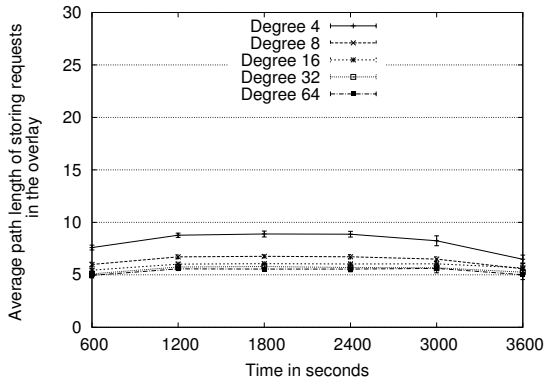

Figure 15. Average path length of the storing requests in the overlay.

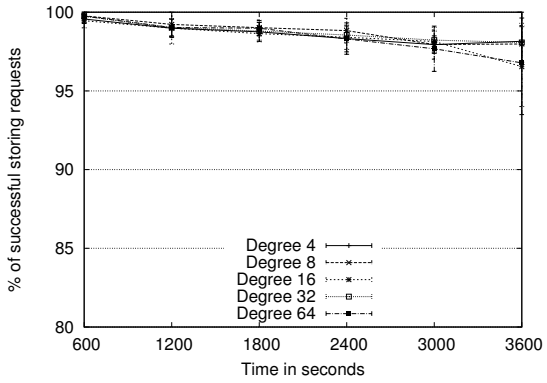

Figure 14. Percentage of successful storing requests.

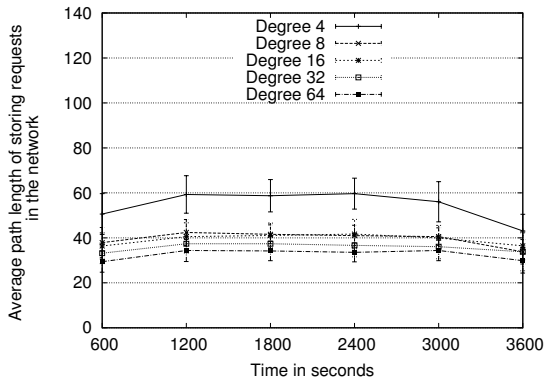

Figure 16. Average path length of the storing requests in the IP network.

the performances of the storing and the solving requests. More precisely we measure the rate of success as well as the average overlay path length of both storing and solving requests.

Figure 14 shows the percentage of successful storing requests over the simulation duration. We assume here that only one copy of a given pair is stored in the system. We can see that given the parameters of the simulation, the rate of success is very high despite the churn.

Figure 15 shows the average path length of the storing requests in the overlay network over the simulation duration. The number of peers to go through including the destination before storing a pair varies from 6 to 9 depending on the addressing tree degree. This number is decreasing when the degree is increasing with a diminishing return effect that can be seen starting at degree 16.

Figure 16 shows the average path length of the storing requests in the IP network over the simulation duration. We can see on this plot that the addressing tree degree has a greater impact on the number of IP hops than on the number of overlay hops. The number of hops are greater of course but also the variability of the values as well as the gaps between the plots of the various degree parameter values are much higher. For a degree of 4 the average hop count is 60 whereas for a degree of 64 the average hop count is around 27. Given the results of Figure 15, we can deduce that the average IP hops between the peers varies from around 4.5 to 6.7 which is lower than the average path length of 7.9 measured in the IPv6 map. We can deduce that the peers which store the bindings are on average closer to the core of the network.

Figure 17 shows the percentage of successful solving requests over the simulation duration. As for the storing request, we can see that given the parameters of the simulation, the rate of success is very high despite the churn.

Figure 18 shows the average path length of the solving requests in the overlay network over the simulation duration. The number of peers to go through to reach the holder of the pair and including the return trip to the sender of the request varies roughly from 9 to 16 depending on the addressing tree degree. A degree of 4 yields a typical path length of 16, a degree of 8 reduces the path length to 12 and degree values above 8 all yield path lengths between 9 and 10. Thus the number of hops is decreasing when the degree is increasing

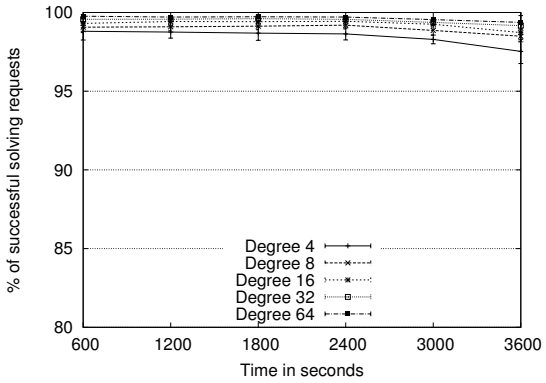

Figure 17. Percentage of successful solving requests.

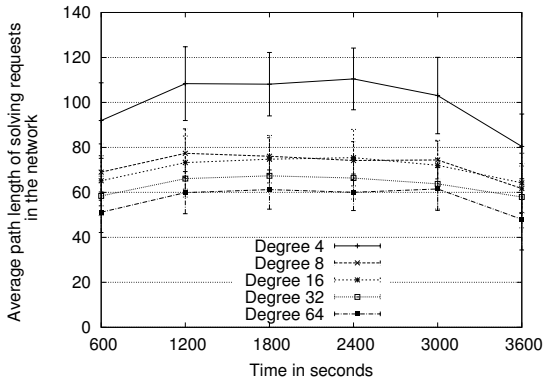

Figure 19. Average path length of the solving requests in the IP network.

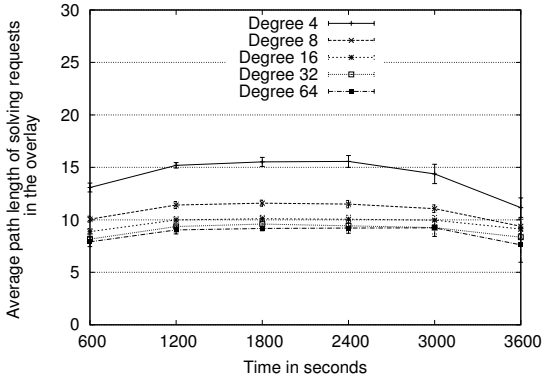

Figure 18. Average path length of the solving requests in the overlay.

with a diminishing return effect around degree 16, similar to the storing requests path lengths of Figure 15.

Figure 19 shows the average path length of the solving requests in the IP network over the simulation duration. We can also see on this plot that the addressing tree degree has a much greater impact on the number of IP hops than on the number of overlay hops. The gaps between the plots of the various degree parameter values are much higher than what was observed for the storing requests in Figure 16. If we except the plots corresponding to the degrees 8 and 16 that are very close to each other, the other plots have widely different path lengths ranging from roughly 110 for degree 4 to 60 for degree 64. Given the results of Figure 18, we can deduce that the average IP hops between the pairs of neighbor peers varies from around 6.6 to 6.9 which is a bit lower than the average path length of 7.9 in the IPv6 map. As a solving request takes a much longer path than a storing request, this explains the reduction in the variability of the number of IP hops between two neighbor peers.

We can conclude that given those simulation results, our overlay routing mechanism remains efficient under dynamics with a success rate above 90%. The average path lengths in the overlay are typically between 15 and 20 IP network hops. Our DHT request shows encouraging performances whatever the degree chosen. The rate of success of both the storing and solving requests is above 95%. The average path lengths of the requests are also acceptable and show typical values for DHT systems.

In order to compare our DHT solution detailed in Section IV to previous existing schemes, we have implemented the addressing, routing and DHT mechanisms of CLOAK inside the `PeerSim` simulator. We have thus obtained comparative simulation results with Chord [14], Kademlia [15] and MSPastry [16] by using the same simulation parameters (e.g., simulation duration, peers' topology, peers' session lengths, etc). We have used an overlay network with a size remaining around 1000 nodes for 2 hours of simulated time. The churn rate varies from 10% to 60% over periods of 10 minutes (i.e., during the 10 minutes, x % of randomly selected peers will leave and be replaced by new ones). Each point on these plots, is the average of 10 runs and the standard deviation is provided.

Figure 20 shows the success ratio of the solving requests as a function of the churn rate. We can see that all DHT schemes perform similarly with a success ratio linearly decreasing with the churn rate. CLOAK has the best success ratio results, closely followed by MSPastry and Chord which have nearly the same values. Kademlia has the lowest success ratio results. As the plots for the storing requests are very similar to the solving ones, we do not show them to avoid redundancy.

Figure 21 shows the average path length measured by hop count of the solving requests as a function of the churn rate. Here again, the DHT schemes have the same behavior with a path length (in hops) slowly decreasing when the churn increases. MSPastry exhibits the shortest path lengths, closely followed by CLOAK. Kademlia has on average 1 more hop than MSPastry whatever the churn, while Chord has the longest path lengths, being on average 2 hops longer than MSPastry and CLOAK, although this difference tends

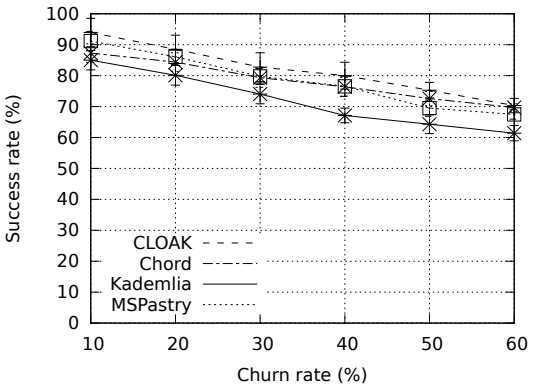

Figure 20. Comparison of the success ratio of the solving requests for various DHT *vs* churn rate.

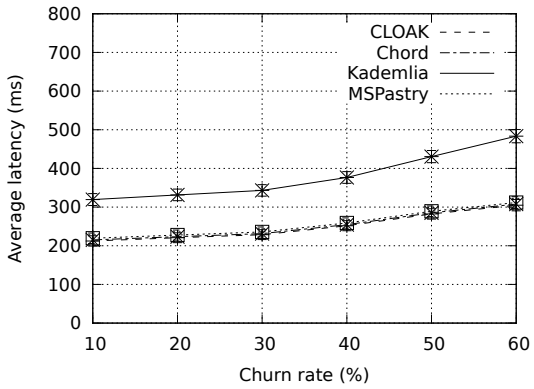

Figure 22. Comparison of the latency of the solving requests for various DHT *vs* churn rate.

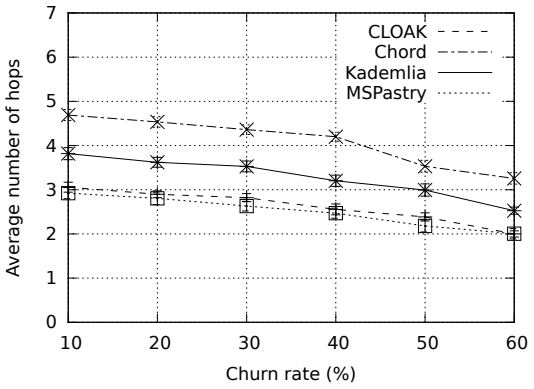

Figure 21. Comparison of the hop count of the solving requests for various DHT *vs* churn rate.

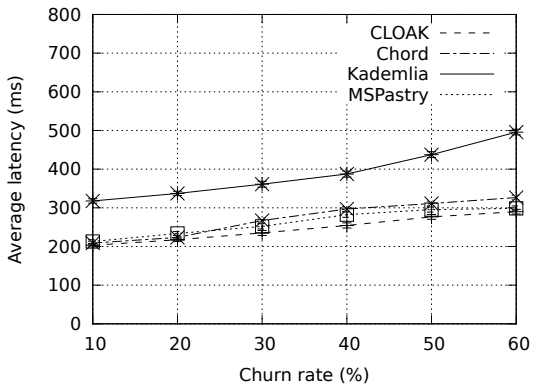

Figure 23. Comparison of the latency of the storing requests for various DHT *vs* churn rate.

to decrease when the churn is equal or above 40%. As for the success ratio, we do not show the plots for the hop count of the storing requests because they are very similar to the solving ones.

Figure 22 shows the average latency of the solving requests as a function of the churn rate. Indeed, as a path length measured in number of hops does not necessarily translate into a higher latency, we have measured the latter one to evaluate the time taken for the requests to complete. All the DHT schemes have nearly the same latency at any churn rate, excepted for Kademlia which is typically 100 ms to 180 ms higher than the others depending on the churn rate. These results illustrate our point above that despite longer path lengths, Chord performs as well as MSPastry and CLOAK when the latency is observed.

Figure 23 shows the average latency of the storing requests as a function of the churn rate. Unlike the success ratio and hop count metrics above, here the plots of the storing requests are a bit different than the solving ones. Starting at 20% of churn, the plots of the latency of Chord, MSPastry and CLOAK begin to separate. CLOAK has the lowest latency values, followed by MSPastry and Chord. However the gaps between these three plots are quite small, typically no more than 25 ms. As above for the solving latency, Kademlia has a storing latency which is typically 100 ms to 170 ms higher than the others depending on the churn rate.

All these comparative results show that CLOAK performs as well as (and sometimes a little bit better than) Chord, MSPastry and Kademlia which are the three popular DHT schemes that we have compared CLOAK to. These results also confirm our analysis presented in Section V. The key advantage of our solution is that peers can connect freely to any other peers they want, while in other DHT schemes such as Chord, peers must insert themselves in the DHT by connecting to other predetermined peers depending on their IP addresses. Another advantage is the cheap cost of building our DHT on top of our addressing and routing system. Using another DHT scheme would impose us to use two different routing schemes with the associated costs.

Our greedy hyperbolic routing scheme for the overlay and a key based routing scheme for the DHT. The simulation results encourage us to keep our own DHT scheme inside the CLOAK overlay.

VII. Related Work

A. Transport Layer Mobility

Virtual connections, as we define them, can be considered as providing (among other benefits) transport layer connection mobility. Research on such transport layer connection mobility has mainly remained experimental up to now. Concerning the TCP connection management, several solutions have been proposed. TCP-Migrate [17], [18] developed at the Massachusetts Institute of Technology, provides a unified framework to support address changes and connectivity interruptions. TCP-Migrate provides mobile-aware applications with a set of system primitives for connectivity re-instantiation. TCP-Migrate enables applications to reduce their resource consumption during periods of disconnection and resume sessions upon reconnection. Rocks [19] developed at the University of Wisconsin, protects socket-based applications from network failures, such as link failures, IP address changes and extended periods of disconnection. Migratory TCP [20], developed at Rutgers University, is a transport layer protocol for building highly-available network services by means of transparent migration of the server endpoint of a live connection between cooperating servers that provide the same service. The origin and destination servers cooperate by transferring the connection state in order to accommodate the migrating connection. Finally, the Fault-Tolerant TCP [21], [22] developed at the University of Texas, ensures a faulty server to keep its TCP connections open until it either recovers or it is failed over to a backup. The failure and recovery of the server process are completely transparent to client processes. However, all these projects only deal with TCP re-connection. They do not provide a total virtualization of the communication and do not permit to switch both applications and/or devices from any communicating user at will. Furthermore, they are based on the domain name and IP address paradigm and do not provide the separation of the naming and addressing planes.

B. Name and Address Separation

Other solutions have been proposed with this separation in mind. However, they are typically placed below the transport layer and require modifications in the host or in the network infrastructure. The Host Identity Protocol (HIP) [23] for example, proposes a new namespace, the Host Identity namespace, and a new protocol layer named HIP, between the internetworking and transport layers. This solution requires the modification of the host stack. Similarly, the Locator/Identifier Separation Protocol (LISP) [24] is a network-based protocol that enables separation of IP addresses into two new numbering spaces: Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). No changes are required to either host protocol stacks or to the *core* of the Internet infrastructure. However LISP requires software changes in *edge* routers and cannot deal with host mobility. LISP can be incrementally deployed and offers traffic engineering and multi-homing benefits even when there are relatively few LISP-capable sites. The Shim6 protocol [25] is a layer 3 shim for providing locator agility below the transport protocols, so that multihoming can be provided for IPv6 with failover and load-sharing properties, without assuming that a multihomed site will have a provider-independent IPv6 address prefix announced in the global IPv6 routing table. Currently, this solution is restricted to the IPv6 network. The Internet Indirection Infrastructure (i3) by Stoica *et al.* [26] is an overlay-based indirection infrastructure that offers a rendezvous-based communication abstraction thus decoupling the act of sending from the act of receiving. Instead of sending a packet to a destination, each packet has an identifier which is used by the receiver to obtain the packet. However, i3 must be defined as a general overlay that everyone should use thus needing third party resources (such as the DNS infrastructure). With CLOAK, we try to enforce the principle that only the members of a given overlay have to share their resources. The Host Identity Indirection Infrastructure (Hi3) by Gurtov *et al.* [27] is a networking architecture for mobile hosts, derived from i3 and HIP. Although Hi3 provides efficient support for secure mobility and multihoming to Internet hosts, we do not adopt this infrastructure in order to avoid the issues of IP stack modifications (HIP) and third party resource requirements (i3). Data-Oriented Network Architecture (DONA) by Koponen *et al.* [28] proposes the use of permanent flat names coupled with name-based routing. Rather than use DNS servers, DONA relies on a new class of network entities called resolution handlers (RHs). As with i3, DONA needs third party resources provided by the infrastructure of RHs.

C. Distributed Hash Table

Concerning the DHT part of our solution, our proposal borrows some elements from well known DHTs. Our mapping mechanism for placing keys on the unit circle is similar to the one defined by Chord [14]. However, unlike Chord we do not place the peers themselves on this circle but inside the unit disk by using complex coordinates. Similarly to CAN [29], we use a multi-dimensional coordinate space, but instead of using a d-dimensional cartesian multi-torus, we use the 2-dimensional hyperbolic plane $\mathbb{H}^2$. Our greedy routing scheme is based on a properly defined distance metric as done in Kademlia [15]. But unlike Kademlia which is based on the XOR metric, we use the hyperbolic distance defined for the Poincaré disk model of the hyperbolic plane. Another advantage of our greedy routing algorithm as opposed to prefix routing algorithms such as those developed

in Pastry [16], is that it does not rely on routing tables. Only the coordinates of the neighbors of a peer are needed to forward a message. This is highly scalable as the peers do not need to build and maintain routing tables. The intuition of using the hyperbolic plane as a virtual address space for our overlay and DHT systems comes from the work of Kleinberg [4]. However, we have defined a novel mapping function, whereas Kleinberg has suggested using CAN for implementing a DHT based on hyperbolic coordinates.

VIII. Conclusion and Future Work

In this paper, we have presented a new architecture called CLOAK designed for providing flexibility to Internet communications by using virtual connections set upon an overlay network. This architecture will be implemented as two protocols running on top of the transport protocols of the devices. The devices using the CLOAK middleware will freely interconnect with each other and thus will form a dynamic P2P overlay network. This overlay will enable the applications to maintain their communications even if some transport layer connections are subject to failures. The middleware will transparently restore the transport connections without killing the connections of the applications. The architecture, by giving identifiers to users and devices, will provide flexibility, security and mobility to applications despite the IP address changes suffered by the devices.

We have also shown that given an appropriate mapping function, it is easy to setup and maintain a consistent DHT structure upon such an overlay. Our theoretical analysis has shown that our DHT proposal is scalable with performances similar to other existing DHTs. We have implemented the overlay addressing and routing part as well as the DHT part of our middleware in our discrete event *nem* simulator as well as in the *PeerSim* simulator and the results are encouraging. Our simulation results have demonstrated that the success rate of the routing procedure, as well as the success rate of the storing and solving requests are typical of such systems. Measurements of path lengths and latencies also confirm the proper behavior of our solution compared to prior ones.

Our future work will be aimed at defining the CLOAK API, implementing the middleware as a library, modifying a relevant test application (such as a chat or video streaming application) and testing it on a virtualized platform for studying the impact of transport layer connection pipelining created by the P2P overlay network.

References

[1] T. Tiendrebeogo, D. Magoni, and O. Sié, "Virtual internet connections over dynamic peer-to-peer overlay networks," in *Proceedings of the 3rd International Conference on Evolving Internet*, 2011, pp. 58–65.

[2] C. Cassagnes, D. Bromberg, and D. Magoni, "An overlay architecture for achieving total flexibility in internet communications," in *Proceedings of the 8th International Conference on Advanced Information Technologies for Management*, 2010, pp. 39–60.

[3] C. Cassagnes, T. Tiendrebeogo, D. Bromberg, and D. Magoni, "Overlay addressing and routing system based on hyperbolic geometry," in *Proceedings of the 16th IEEE Symposium on Computers and Communications*, 2011, pp. 294–301.

[4] R. Kleinberg, "Geographic routing using hyperbolic space," in *Proceedings of the 26th IEEE INFOCOM*, 2007, pp. 1902–1909.

[5] Rosenberg, Schulzrinne, Camarillo, Johnston, and P. et al., "SIP: Session initiation protocol," Internet Engineering Task Force, Request For Comments 3261, June 2002.

[6] ITU-T, "H323: Packet-based multimedia communications systems," ITU-T, Recommendation, December 2009.

[7] J. Postel and J. Reynolds, "File transfer protocol (FTP)," *Internet Engineering Task Force*, Request For Comments 959, 1985.

[8] G. Wright and R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.

[9] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, pp. 72–93, 2005.

[10] A.-L. Barabási and E. Bonabeau, "Scale-free networks," *Scientific American*, vol. 288, pp. 60–69, 2003.

[11] R. Cohen and S. Havlin, "Scale-free networks are ultrasmall," *Phys. Rev. Lett.*, vol. 90, no. 5, p. 058701, Feb 2003.

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

[13] D. Magoni and M. Hoerdt, "Internet core topology mapping and analysis," *Computer Communications*, vol. 28, no. 5, pp. 494–506, 2005.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 149–160.

[15] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer-Verlag, 2002, pp. 53–65.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001, pp. 329–350.

[17] A. C. Snoeren, H. Balakrishnan, and M. F. Kaashoek, "Reconsidering IP mobility," in *Proceedings of the 8th HotOS*, 2001, pp. 41–46.

[18] A. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," in *Proceedings of the 6th ACM MobiCom*, 2000, pp. 155–166.

[19] V. Zandy and B. Miller, "Reliable network connections," in *Proceedings of the 8th ACM MobiCom*, 2002, pp. 95–106.

[20] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: Connection migration for service continuity in the internet," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002, pp. 469–470.

[21] D. Zagorodnov, K. Marzullo, and T. Bressoud, "Engineering fault tolerant TCP/IP services using FT-TCP," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, 2003, pp. 393–402.

[22] T. Bressoud, A. El-Khashab, K. Marzullo, and D. Zagorodnov, "Wrapping server-side TCP to mask connection failures," in *Proceedings of the 20th IEEE INFOCOM*, 2001, pp. 329–338.

[23] P. Nikander, A. Gurtov, and T. Henderson, "Host identity protocol (hip): Connectivity, mobility, multi-homing, security, and privacy over ipv4 and ipv6 networks," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 186–204, 2010.

[24] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/identifier separation protocol," *Internet Engineering Task Force*, Internet Draft, July 2011.

[25] E. Nordmark and M. Bagnulo, "Shim6: Level 3 multihoming shim protocol for IPv6," *Internet Engineering Task Force*, Request For Comments 5533, June 2009.

[26] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of ACM SIGCOMM'02*, 2002.

[27] A. Gurtov, D. Korzun, A. Lukyanenko, and P. Nikander, "Hi3: An efficient and secure networking architecture for mobile hosts," *Computer Communications*, vol. 31, pp. 2457–2467, 2008.

[28] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '07, 2007, pp. 181–192.

[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 161–172.