Virtual Internet Connections Over Dynamic Peer-to-Peer Overlay Networks

Telesphore Tiendrebeogo, Damien Magoni
University of Bordeaux – LaBRI
Bordeaux, France
{tiendreb,magoni}@labri.fr

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

***Abstract*—Current Internet applications are still mainly bound to the state of 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 communications from their supporting devices. This enables the complete separation of the devices, applications and users. Our architecture is based on a peer-to-peer overlay network that provides its own distributed hash table system. Preliminary simulation results show that our proposal is feasible.**

***Keywords*-Overlay; virtual connection; distributed hash table.**

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 equipments themselves (e.g., computers, smart phones, routers, etc.) and not on the logical part of the communications. In fact, although we already have a lot of mobile equipments, 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 P2P overlay networks built on top of the TCP/IP protocol stack of the participating devices. We have called this architecture CLOAK (Covering Layers Of Abstract Knowledge). This architecture supports names for entities (i.e., users) and devices, virtual addresses for devices and logical sessions that enable a full virtualization of all kinds of Internet communications. The new semantics brought by our proposal open up many novel possibilities for Internet communications. The virtual connections setup and managed by our solution enable for instance the transparent handling of the breakdown and restore of transport layer connections (e.g., such as TCP or SCTP connections).

The remainder of this paper is organized as follows. Section II outlines the related previous work done on virtual connections. Section III presents the design and features of our architecture. Section IV describes its implementation. Section V presents some preliminary results obtained by simulations. Finally, we conclude the paper and present our future research directions.

II. Related work

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 [1], [2] developed at the Massachusetts Institute of Technology, provides a unified framework to support address changes and connectivity interruptions. Migrate provides mobile-aware applications with a set of system primitives for connectivity re-instantiation. Migrate enables applications to reduce their resource consumption during periods of disconnection and resume sessions upon reconnection. Rocks [3] developed at the University of Wisconsin, protect sockets-based applications from network failures, such as link failures, IP address changes and extended periods of disconnection. Migratory TCP [4], 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 [5], [6] developed at the University of Texas, allows 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 enable the total virtualization of a communication. They also do not allow to switch both applications and/or devices from any communicating user at will.

III. 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 enable 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 enables a communication to have 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.

Fig. 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 Fig. 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 Fig. 1) during a communication and all the movements and changes of devices, applications and entities are supported. Note that in Fig. 1, only one instance of each part (device, application, entity) of a communication is shown, other instances would obey the same scheme.

B. Operation

In order to provide all the above mentioned features, our architecture sets up and maintains a P2P overlay network. All the devices that wish to share resources in order to benefit from the architecture join together to form an overlay. Fig. 2 shows an overlay example with the links shown in dotted red lines. The devices (i.e., end-hosts) connect to the others by creating

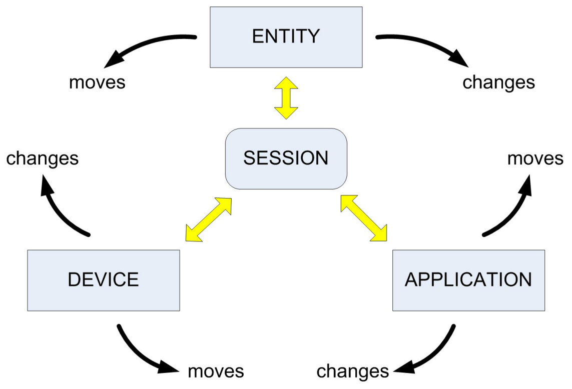

Figure 1. CLOAK communication paradigm.

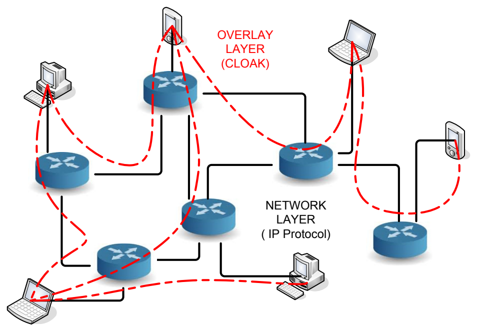

Figure 2. Overlay network.

virtual links (i.e., transport layer connections). Devices with two or more links play the role of overlay routers. We allow the overlay network to build up without any constraints. Network devices can connect arbitrarily to each other and join and leave the P2P network at any time.

When joining the overlay, each device obtains a unique overlay address. The method for addressing the peers and routing the packets inside the overlay is based on the groundbreaking work of Kleinberg [7] that assigns addresses equal to coordinates adequately taken from the hyperbolic plane (represented by the open unit disk). This method creates a greedy embedding of an addressing tree upon the overlay network. This addressing tree is a regular tree of degree k. However in Kleinberg's proposal, the construction of the embedding requires a full knowledge of the graph topology which is also considered static. This is required as the degree k of the addressing tree is equal to the highest degree found in the network. We have enhanced his proposal in order to manage a dynamic topology which is able to grow and shrink over time. Indeed, as 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 [7], 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 enable the use of a greedy routing based on the hyperbolic distance metric that is guaranteed to work. 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. Our dynamic method is fully described in our previous paper [8].

In order to set up the DHT (Distributed Hash Tables) 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.
5) Invite in this session another entity to communicate with.
6) Set an overlay layer virtual connection to this entity as shown in Fig. 3.
7) Send the data stream through this connection.

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 should be attributed a unique identifier that defines the session during its lifetime in the overlay.
- Device namespace: any device should be attributed a unique identifier that permanently represents the device. The lifespan of this identifier should be as long as the lifespan of its corresponding device.
- Entity namespace: any entity should be attributed a unique identifier that represent 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 should be as long as the lifespan of its corresponding entity.
- Application namespace: any application used during a part or all of a session should be attributed a unique identifier that enables it to receive data from the other applications of this session. The lifespan of this identifier should be equal to the lifespan of the use of the application. If the entity switches to another application, this identifier should be updated.

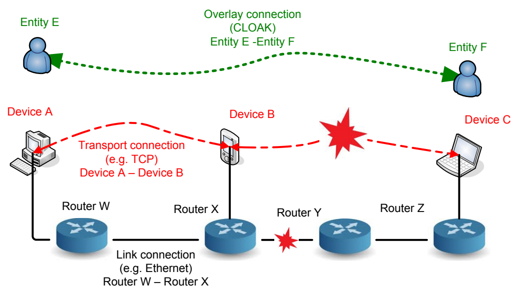

Figure 3. Virtual connections.

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. Fig. 4 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 will enable the survival of the session until all the entities involved decide to stop it. Fig. 5 shows a typical scenario relying on this session management

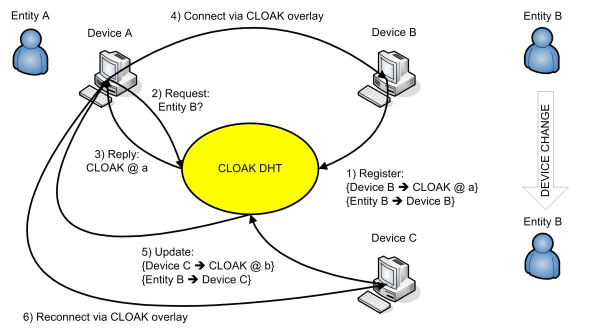
Entity A
4) Connect via CLOAK overlay
Entity B
Device A
Device B
2) Request: Entity B?
3) Reply: CLOAK @ a
CLOAK DHT
1) Register: {Device B ➔ CLOAK @ a} {Entity B ➔ Device B}
DEVICE CHANGE
Entity B
Device C
5) Update: {Device C ➔ CLOAK @ b} {Entity B ➔ Device C}
6) Reconnect via CLOAK overlay

Figure 4. Identification and localization.

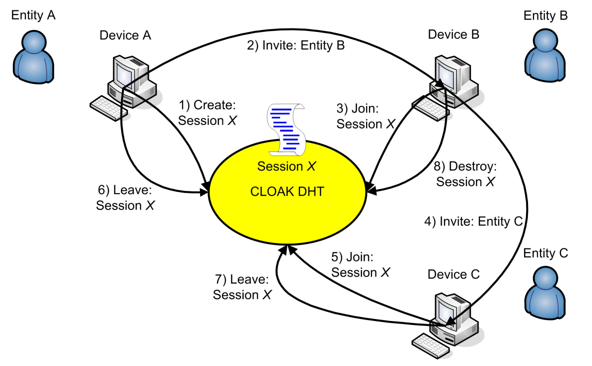

Figure 5. Session management.

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 leave the session X allowing 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 enables the definition and use of new namespaces. It can build scalable and reliable dynamic Virtual Private Networks, define fully isolated Friend-to-Friend networks, serve as an anonymizing layer for Darknets 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-time audio and video communications based on protocols such as SIP and H323. Sharing applications encompass file-sharing, blogging and

TABLE I
FEATURES FOR *cloaked* APPLICATIONS.

Application type	Messaging	Conferencing	Sharing	Streaming
Reachability	✓			
Mobility		✓		✓
E2E privacy	✓	✓		
E2E authentication	✓	✓		
Anonymity			✓	✓
Redirection	✓			✓
Multicasting		✓	✓	✓

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 now 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 uses security by using entity IDs, thus establishing End-to-End (E2E) privacy and authentication. 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 anonymity. Redirection is the ability to forward a message or a stream to another entity. Finally, multicasting support is provided by CLOAK as group addresses can be easily set up in the DHT. This feature is useful for saving bandwidth during group communications.

IV. Implementation

Fig. 6 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 for these new protocols. These new identifiers enable data streams to be bound by virtual identifiers instead of the typical network identifiers (i.e., IP address, protocol n°, port n°) that are now able to change without breaking the communication. As shown in Fig. 1, identifiers of devices, applications and entities are interwoven together inside a session, but 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

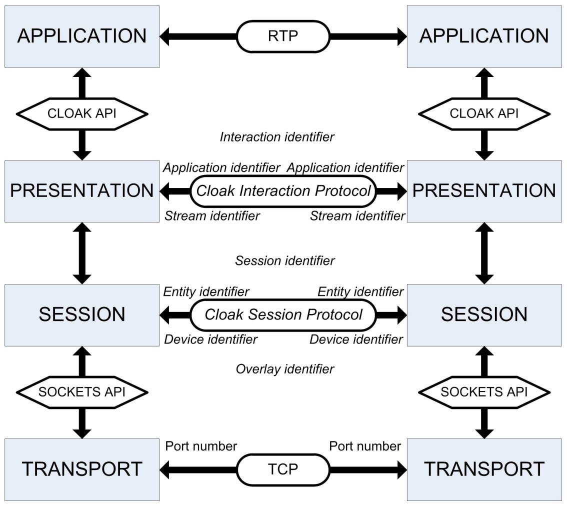

Figure 6. CLOAK architecture in the OSI model.

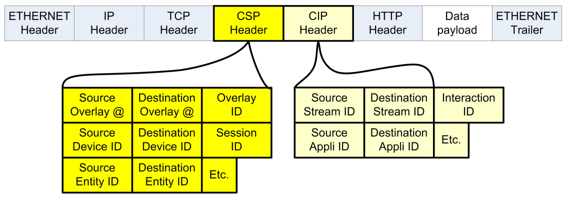

Figure 7. CLOAK protocol encapsulation.

transport layer protocols. Thus a file transfer (FTP [9]) 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 [10]. 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 Fig. 6 how the CLOAK architecture fits in the network protocol stack. We will show now how this design translates into the format of the packet headers. Fig. 7 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 allow for virtual port numbering on top of the entity. The application identifiers allow for selecting or switching applications when it makes sense in a communication.

The definition and implementation of the CLOAK additional protocols (CSP and CIP) and their corresponding headers enable our architecture to solve 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 firewall issues because any type and any number of transport layer connections can be used to connect 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 application ID fields that enable numerous applications to be multiplexed on a single transport connection if necessary. CLOAK also solves security issues because the security protocols can create security associations by using entity identifiers instead of IP addresses. The security is then by design independent from the devices and applications involved.

Fig. 8 shows the modules composing the CLOAK middleware. We can see that many are needed to enable the proper functioning of the CLOAK architecture. The functionality provided by each module is briefly described below:

- 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 greedily routing the overlay packets with 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.
- Interact: primitives for managing the bindings between the data streams and the applications.

For not overwhelming the paper with too much details, the functioning of the bootstrap, steering and interacting modules will be done in a future work.

V. 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* [11] for obtaining all the results shown in this paper.

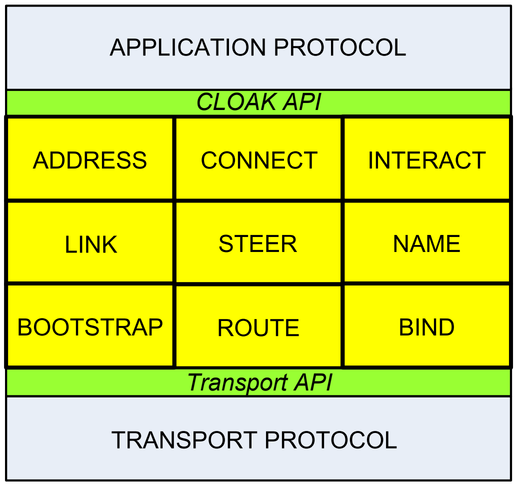

Figure 8. Modules of the middleware.

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 [12]. In all simulations, the first peer creating the overlay is always a randomly picked node of the map. We have considered that only some part of the 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 simulations, 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 of the graphs shown below. Indeed, 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.

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.

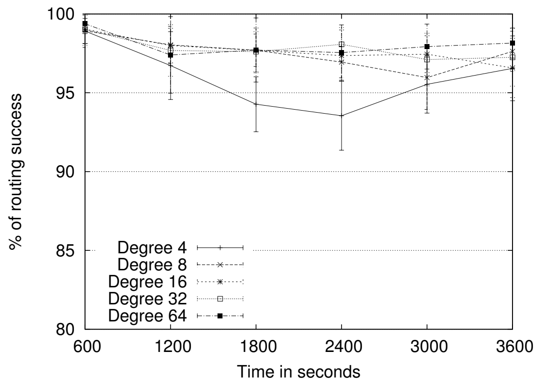

Figure 9. Average routing success rate

B. Results

We evaluate here the performances of the overlay routing depending on the chosen fixed addressing tree degree as explained in III-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 Fig. 9, 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.

Fig. 10 shows the average path length of the hyperbolic routing. The path length is measured as the number of IP hops covered by the packet from the source peer to the destination peer. We can see that values are larger than the ones measured in the static simulations because here only a subset of the nodes are peers belonging to the overlay thus statistically increasing the distances. In the static simulations, the paths from all pairs were evaluated and the overlay topology was the same as the map itself. Here the nodes form an overlay which may have a different topology and thus lower path length optimality. This remains true even though overlay peers always try to establish overlay links to hop-wise closer peers.

Fig. 11 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 ex-

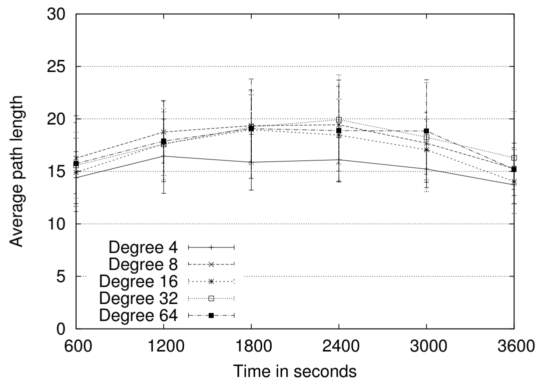

Figure 10. Average path length between peers

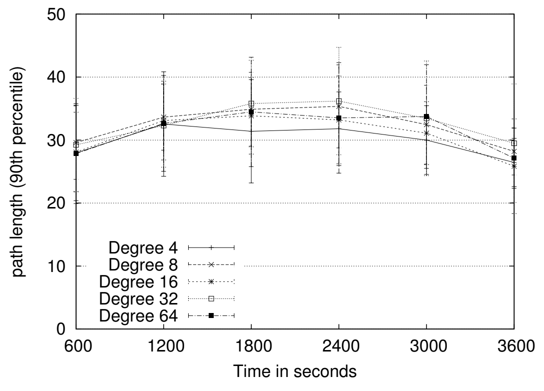

Figure 11. 90th percentile path length between peers

treme 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 fig. 10. We conclude that including the values from the median to the 90th percentile yields a path inflation of 100% which is important but still bearable.

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 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.

Fig. 12 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.

Fig. 13 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.

Fig. 14 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.

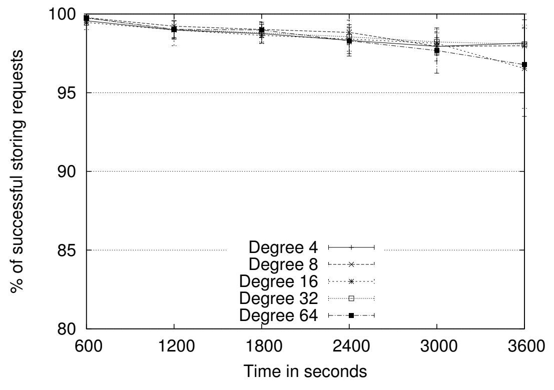

Figure 12. Percentage of successful storing requests.

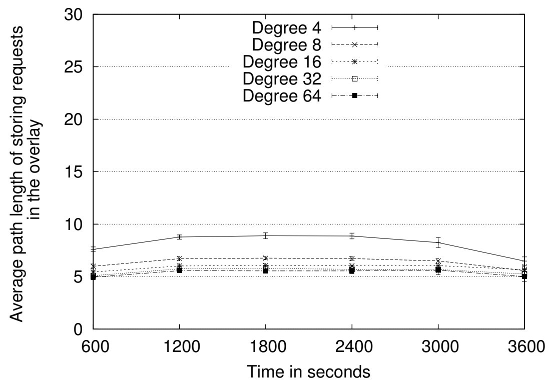

Figure 13. Average path length of the storing requests in the overlay network.

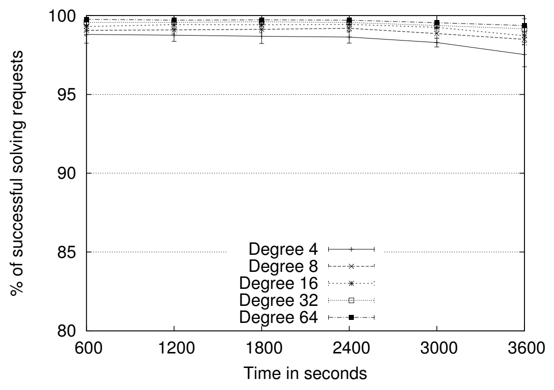

Figure 14. Percentage of successful solving requests.

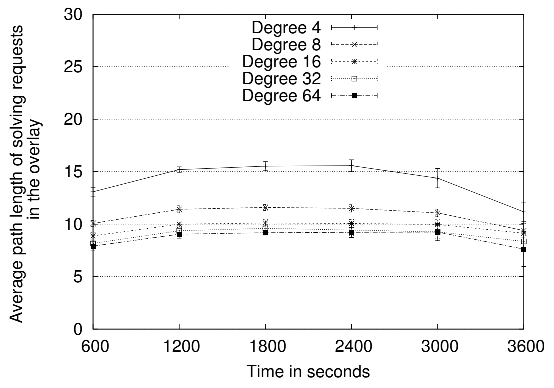

Figure 15. Average path length of the solving requests in the overlay network.

Fig. 15 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 with a diminishing return effect around degree 16, similar to the storing requests path lengths of Fig. 13.

We can conclude that given those simulation results, our DHT shows encouraging performances whatever the degree chosen. The rate of success of both the storing and solving requests, for an overlay running for one hour with a total of 1800 peers, is very high. The average path lengths of the requests are also acceptable and show typical values for these kind of systems.

VI. Conclusion

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 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 implemented the overlay addressing and routing part as well as the DHT part of our middleware in a simulator and preliminary results are encouraging.

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 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] A. C. Snoeren, H. Balakrishnan, and M. F. Kaashoek, “Reconsidering ip mobility,” in *Proceedings of the 8th HotOS*, 2001, pp. 41–46.

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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*, to appear, 2011.

[9] J. Postel and J. Reynolds, “File transfer protocol (ftp),” Request For Comments 959, 1985.

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

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

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