Design and implementation of a hierarchical SIP-based peer-to-peer network

Oscar Bravo, Antonio Costa and Maria João Nicolau
Centro Algoritmi, Universidade do Minho, Portugal
Email: a48058@alunos.uminho.pt, costa@di.uminho.pt, joao@dsi.uminho.pt

Abstract—**Peer-to-peer (P2P) has gained popularity in Internet due to the increased number of users and distributed services such as file sharing and voice calls over IP (VoIP). Currently, the most popular P2P networks store information pertaining to its resources in a distributed manner using Distributed Hash Tables (DHT). In this type of networks, the peers are deterministically positioned and resources are also allocated to each peer for indexing in a deterministic manner.**

This paper presents a new hierarchical SIP based P2P network. A pure SIP solution was used because we believe that an open solution such as SIP can facilitate the creation of new types of services and permit the facilitated integration of different services. A two level hierarchy is used and is aimed towards the improvement of network efficiency where peers can move dynamically from one layer to another according to its available resources. In addition to this, the proposed implementation architecture allows the independence of the underlying DHT algorithms.

The proposed architecture was implemented and tested in a realistic scenario which was created inside a Linux cluster. The DHT algorithms, Chord and EpiChord, were also implemented and used to index resources in either flat or hierarchical networks. Results indicate that the proposed two-layer hierarchy significantly improves the P2P overlay performance while in the presence of peers with limitations.

Index Terms—**P2P Networks, DHT Algorithms, P2PSIP, SIP**

I. Introduction

The increased number of users on the Internet in combination with the large number of available services, makes peer-to-peer networks assume an increasingly important role. Currently, there is a large amount of scientific work that addresses various aspects of these types of networks. The P2P network scalability is one of the most important aspects to consider because these networks increasingly consist of a large number of nodes which often has a direct influence on P2P network performance.

In a traditional P2P overlay, all peers have the same role and importance. They are equally involved in the routing, storage and location of resource tasks. However, there are cases where assigning the same importance to all peers may not be the most desirable for either performance reasons or for other reasons such as security issues. This work presents a particular interest in studying the influence that nodes possessing lower capacities can have on the performance of the P2P overlay.

In this context, we consider that a member may have a negative impact on P2P network performance for several reasons: poor quality network connection (traffic limitations, lower bandwidth, or links with a high probability of packet losses), little time spent in the overlay which can generate a large amount of traffic to maintain the overlay (transfer of resources, updating tables, etc) and limited resources in terms of hardware (CPU, memory and battery). Smart-phones are a good example of a mobile device which is able to participate in a P2P overlay however, its participation may have a negative impact on the overlay performance due to possible limitations in terms of connectivity and its characteristics, including the battery.

The existence of peers in the overlay along with some of the listed conditions, may influence the performance of overlay which will then increase the time needed to locate resources. In order to minimize these problems, a two-tier P2P overlay was implemented. The lowest level of the hierarchy is composed of nodes with some limitations called clients. A client does not form or participate in P2P overlay, does not receive messages that are not for it, nor is it responsible for storing data. Instead, it uses a peer belonging to the overlay as an intermediary to access the overlay services. The upper level of the hierarchy consists of nodes (peers) that actually form the P2P overlay. They act as normal peers by storing resources and forwarding messages between themselves. The only change that is needed on peers, is the addition of support for clients. That is, allow peers to receive requests to store or find resources in the overlay from outside it. Whenever a request from a client is received, for example the storage of a resource in the overlay, the peer receiving the request from the client stores the resource in the overlay as its own.

Many of the existing solutions that aim to implement peer-to-peer networks are closed solutions designed for specific applications. However, there is an academic effort which aims to develop generic peer-to-peer networks, based on open solutions. The use of generic networks possesses some advantages because they are not limited to one type of service or application. Instead, they can be used to implement different services or support multiple applications. Moreover, the fact that they use open solutions such as SIP (Session Initiation Protocol) [1], may also represent an advantage because there is a greater knowledge of how these solutions work which can facilitate its implementation as well as allow different entities to gain improved interoperability with each other since the operation of the solution used is well known.

In order to evaluate the hierarchical P2P network, a JAVA implementation was developed. Communication is attained through a P2PSIP protocol and, of the various solutions for P2PSIP protocols studied, the draft specification protocol dSIP [2] was chosen because it is entirely based on SIP. A new message type is proposed in order to distinguish between clients and peers and support the two level hierarchy. Two DHT algorithms were also implemented: Chord [3] and EpiChord [4]. Due to its popularity, Chord is mandatory in all solutions. EpiChord is a variant of Chord and was also chosen as of a result of its claims to improve the overlay performance. The Chord implementation was based on the work of Cyrano [5] although many of the components have been adapted and rewritten. The implementation of EpiChord was built from scratch. Chord and Epichord were extended in order to support the hierarchical structure of the proposed P2P overlay.

This paper is structured as follows: section II describes peer-to-peer networks and emphasizes major related work on P2PSIP approaches. This section also includes a description of Epichord. Section III presents a conceptual description of the proposed hierarchical SIP based solution and explains major decisions taken. Section IV provides details regarding the Java implementation built. Section V describes a set of tests conducted in a realistic emulated network environment, for flat and hierarchical overlays, in the presence of limited peers. Results are then compared and discussed. Finally, section VI presents conclusions and outlines future work.

II. Peer-to-peer Networks

In a peer-to-peer network, the manner in which data is indexed as well as how the nodes (peers) are positioned in the network (overlay), has lead some authors to classify P2P networks into two distinct forms: unstructured and structured overlays [6] [7]. In an unstructured P2P overlay, the mechanism for resource location on the network usually involves the use of flooding techniques [6] [7]. Here the network is flooded with messages aiming to find the location of the desired resource. This mechanism presents some issues due to the large amount of traffic that can be generated. In addition to this, it cannot ensure that a resource existing somewhere in the network will be found. Currently, P2P networks are mostly structured networks. Structured overlays are characterized by the fact that its topology is predefined and nodes are placed in the overlay in a controlled manner. Resources are also allocated to peers in a deterministic way. This improves resource location and makes it more efficient. This type of overlay is sometimes called a DHT overlay because the overlay usually relies on mechanisms based on Distributed Hash Tables (DHTs) for the placement of resources. Currently, the most popular DHT algorithms are Chord [3] and Kademlia [8].

A. SIP based Peer-to-peer Networks

SIP (Session Initiation Protocol) is a standard signaling protocol created in the context of IETF (Internet Engineering Task Force) which works at the application layer. It is widely used for establishing sessions between one or more participants, for example; in the establishment of telephone calls over the Internet, multimedia content distribution, conferences, etc. Currently, there is an IETF working group (P2PSIP WG) dedicated to the study and creation of P2PSIP protocols. Of the various existing proposals, dSIP [2], P2PP [9] and RELOAD [10] were studied. The P2PP [9], is a draft specification for a P2P protocol which allows the P2P overlay to be created through a structured or unstructured P2P protocol. The messages used by this protocol are not SIP messages, but binary messages instead. With this protocol, sending SIP messages between peers is achieved by encapsulating SIP messages in binary P2PP messages. The RELOAD protocol (Resource Location And Discovery) [10] [1] is the latest proposal from the P2PSIP WG. Just as P2PP, it also uses a binary protocol for the construction and management of the overlay, allowing the use of SIP or other protocols on top of RELOAD protocol.

B. Distributed Session Initiation Protocol (dSIP)

Distributed Session Initiation Protocol (dSIP) is a draft specification [2] of a P2PSIP protocol. It is regarded as an evolution of SoSIMPLE [11] which was specified by some of the same authors of dSIP. This protocol is based entirely on SIP which is used to perform the entire management of the P2P overlay such as peers and resource management.

The fact that SIP is an extensible protocol permitted the creation of new SIP headers needed to carry relevant information to the management of the P2P overlay. According to the authors of dSIP, the use of traditional SIP messages, allows the use of mechanisms commonly used in SIP, in dSIP in order to overcome the problems caused by NATs and firewalls.

The dSIP protocol was developed to be modular and can be used with multiple DHT algorithms, requiring support at least for Chord. Peers are organized in overlay according to the DHT algorithm in use. Unique identifiers (*Peer-ID* and *Resource-ID*), are assigned to peers and resources, both of which must belong to the same address space. The calculation of these identifiers can be obtained using different hashing algorithms however, all overlay peers must use the same algorithm. For example, identifiers can be obtained by applying the SHA-1 algorithm to the combination of IP address and port of the peer or by using a certifying entity responsible for issuing the identifiers. The DHT algorithm uses *Peer-ID* to determine the location of the peer in the overlay as well as the identifiers of the resources for which the peer is responsible. The *Resource-ID* is used to identify resources and can be obtained by applying a hash function on the name or on a set of words that describe the resource. The resource is stored in the peer that has the *Peer-ID* which is closest to the *Resource-ID*.

Due to the constant entry and exit of peers into and out of a P2P network, the information regarding resources stored in the overlay must be constantly exchanged between peers so that resources are always accessible. Data redundancy mechanisms are implemented in order to prevent data loss when a peer fails before transmitting information on resources for which it was responsible to another peer.

The precise manner in which the location of resources is done depends on the DHT algorithm used. Generally speaking, for the location of a given resource, a peer should consult its routing table and send the message to the peer that has the *Peer-ID* nearest to the *Resource-ID* of the wanted resource. This peer, depending on the routing mechanism in use, should forward the message to the closest known peer or send a reply message containing this information.

C. EpiChord

EpiChord is a resource location algorithm for peer-to-peer networks based on Distributed Hash Tables (DHTs), developed by Ben Leong et al [4]. One of the main features that differentiates EpiChord from other existent DHT algorithms is the use of a routing table (called by the authors as cache) with no maximum limit of entries. Other DHT algorithms such as Chord have a maximum of O (log N) entries. Since the number of entries in the routing table can affect the number of hops needed to find a resource in the overlay, EpiChord authors claim that this reduces the average number of hops needed to locate a resource.

1) Routing Strategy: In addition to the use of a cache with no defined size, the EpiChord implements a new strategy known as reactive routing. This routing strategy uses the resource location messages to piggy-back useful information to maintain the cache of the overlay nodes. The nodes in the overlay observe the resource location traffic that they receive and add routing information to the messages they send in order to keep nodes' caches updated. On the other hand, other DHT Algorithms such as Chord, need to periodically send messages to the various entrances of their routing tables (finger table in Chord's case) in order to verify their validity. In EpiChord, this behavior is only necessary if network traffic is too low. In this case, the number of messages that each node receives may not be enough to keep the cache updated. If necessary, an EpiChord peer sends messages to only a few entries from its cache in order to keep the cache updated.

With this strategy, the information contained in the cache is not always the most up to date when compared to other routing strategies. Therefore, in order for resource location to occur, a mechanism that sends messages in parallel to minimize the effect of invalid entries in the cache will be necssary. Sending messages in parallel is not always a good solution because of the extra traffic it generates. However, being as the EpiChord cache can have a very large number of entries, the number of hops needed to find the resource is reduced which in turn causes a reduction of the generated traffic. The Epichord authors state that sending messages in parallel can improve the average performance of the resource location in number of hops and latency when compared to a traditional implementation of Chord with similar resource location traffic.

2) Resource Location: As stated above, the location of resources in EpiChord is completed by sending messages (called queries) in parallel. To initiate the location of a resource, p messages are sent, where p is a configurable parameter of the system.

The algorithm used to determine the destination for each of the p outgoing messages is simple: the cache is consulted in order to obtain a set of nodes closer to the node responsible for the resource to locate. For example, by calling id the identifier of the resource to locate, the p messages are sent as follows: a message is sent to the node in the cache that is the immediate successor of the id to locate and $p-1$ messages are then sent to nodes (from cache) that precede the id of the resource. Figure 1 shows an example of the algorithm described above, the node x, in this example, wants to find the resource identified by id.

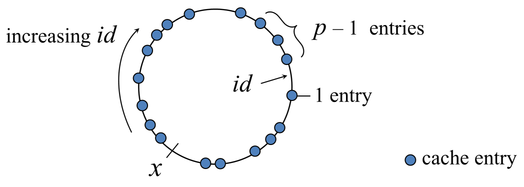


Fig. 1. Lookup Example - Destination of **p** messages [4]

When a node x initializes the location of a resource identified by id, the nodes that receive messages for the resource location must send a reply message according to the following conditions:

- If the node is responsible for the resource, the answer must contain the value associated with the resource (if any) and it must also send routing information containing the details of its predecessor.
- If the node is the predecessor of id relatively to node x, the response message contains information regarding the node's successor as well as the best l next hops obtained through the cache, where l is a configurable parameter of the system.
- If the node is the successor of id relatively to node x, the response message should contain information as to the node's predecessor as well as the best l next hops obtained through the cache.

When the responses are received by node x, it checks the routing information contained and updates its cache with this information. In addition to updating the cache, x sends a new location message to the nodes contained in the response messages that are closer to the destination compared to nodes that have already responded. That is, as peers respond, the two closest nodes (successor and predecessor of the identifier to locate) are updated. New messages are only sent if the recipient is closer to the destination than the current best successor or predecessor.

The location algorithm uses an iterative routing strategy which allows the node that initiates the location to verify if it is approaching the desired node as well as to keep track of sent messages. It is possible to detect cases where a node is referenced multiple times in response messages from other nodes, thus avoiding sending repeated messages to the same node.

III. P2P Hierarchical Overlay

Although the solution presented is an open and generic P2P overlay that suits multiple applications, the developed prototype was intended to support a VoIP application. Therefore, once the overlay is formed, it is used to store and locate resources, providing in a distributed manner, services that usually a Registrar and a Location Server offer in a traditional VoIP architecture.

The resources that are stored by a P2P overlay consist in a key/value pair where the key is the identifier of the resource and the value is the resource itself. In the context of this work, the key is the SIP address of the user and the value is the current temporary SIP location where the user can actually be contacted. The resource identifier is used by the DHT overlay algorithm to define the location in which the resource must be stored in the overlay.

dSIP was the chosen P2PSIP protocol to use due to the fact that this is a solution based entirely on SIP. The fact that dSIP is entirely based on SIP is advantageous because the SIP protocol is standardized, supported natively in many devices, and widely used in various applications including voice calls over IP (VoIP). The use of a well-known protocol such as SIP, makes it easier to implement new services as well as allow interoperability between different services.

A. Two-tier Hierarchy

In addition to the creation of a P2PSIP overlay using the Chord or EpiChord, the application is able to create an overlay with two hierarchical levels. The first hierarchical level is composed exclusively of peers. The second level contains clients that connect to one or more peers and uses services provided by the overlay without actively participating in its construction and management. The creation of this hierarchy is based on the idea that in some scenarios, it may be better for the performance of the overlay that certain peers do not actively participate in the overlay, becoming clients. For active participation in the overlay, one can understand the exchange of messages between peers, the management and construction of the overlay, and also the storage of information. Figure 2 shows an example of the hierarchical structure described. The figure depicts that an overlay is composed exclusively of peers. Clients are in the lower level accessing overlay services through one or more peers.

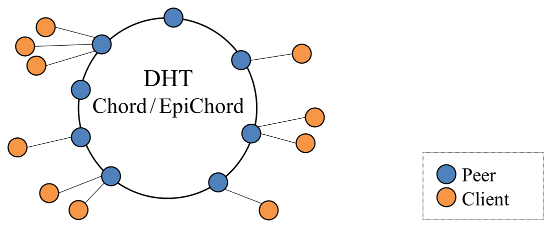


Fig. 2. P2PSIP - Two-tier Hierarchy

B. DHT Algorithms

The draft specification of the dSIP [2] protocol does not impose restrictions on DHT algorithms that can be used by the overlay. However it states that at least the Chord algorithm must be implemented. In this work, in addition to the mandatory implementation of Chord algorithm, it was decided that the EpiChord be implemented as well. This is a variant of the Chord algorithm which promises to improve the overall overlay performance. With the implementation of these two DHT algorithms, we intend to analyze the performance of both in a real scenario in order to verify whether or not the results obtained are in agreement with the simulation results obtained by the authors of EpiChord. In addition to this, we also intend to analyze its usage in a hierarchical overlay.

In the application developed in this paper, both DHT algorithms use SHA-1 with 160 bits as the hashing algorithm used to generate the hash of the identifiers for peers and resources. The use of identifiers consisting of 160 bits means that the ring formed by the peers in the overlay has an address space of 2^{160} identifiers. Another important aspect in the implementation of the DHT algorithms is the routing strategy. Due to the fact that the EpiChord algorithm was designed to work with an iterative routing strategy, it was decided that this should be the behavior of both algorithms: Chord and EpiChord.

C. SIP messages used

The messages used by dSIP protocol messages are traditional SIP messages with the addition of two new types of headers (*DHT-PeerID* and *DHT-Link*) needed to carry information relevant to the P2P overlay management.

Since this work is intended to have clients and peers in a two-level hierarchy and as dSIP protocol specifies only messages that should be exchanged between peers, it was necessary to specify a new type of message to be exchanged between clients and peers. To simplify this process, a new SIP header called *ClientID* was created. It is based on the *DHT-PeerID* header. The new header is only used by clients and aims to allow a peer to identify the origin of a message. The existence of a header *ClientID* or *DHT-PeerID* in a message allows peers to easily identify whether or not the message is coming from a client or a peer.

IV. Implementation

This section describes a few aspects of the java implementation of the proposed peer-to-peer network. Figure 3 ilustrates the software architecture of the implementation. It was designed to be generic and modular so that it can be used by multiple applications and include other DHT algorithms. In order to achieve this, the components are structured into 4 stacked layers.

The two bottom layers (named SIP and Transport) are the basic communication layers which ensure a standard SIP protocol on top of TCP or UDP. They were implemented using the JAIN-SIP [12] library. This library provides a set of APIs that allow you to send and receive SIP messages. Since the upper layers only use a simple subset of the API, it could

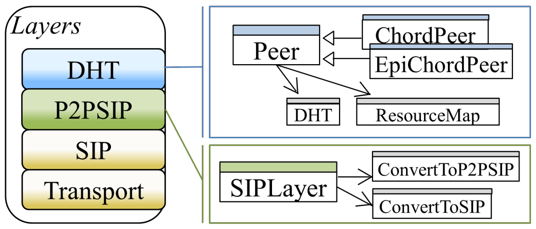


Fig. 3. Application architecture - Layers

be replaced by another SIP stack implementation if necessary. The two upper layers (DHT and P2PSIP) are responsible for the creation and maintenance of the P2P overlay using SIP. They are described in more detail in the next sections.

A. P2PSIP Layer Implementation

The P2PSIP layer is an intermediary layer that further abstracts the SIP communication to the upper layer. It provides a simple and more adequate high level communication API to the DHT layer. It hides the details of the SIP supporting stack and breaks dependencies between DHT and the SIP supporting library. The SIP layer could be replaced in a more facilitated manner.

As shown in figure 3, the *SipLayer* class is the main class for this layer. The main interface for the upper layer consists of two simple methods which can be invoked to send and receive P2PSIP messages:

- *public void sendP2PSIPrequest(P2PSIPMessage msg, ITransactionListener callback)* - This method is used for sending a P2PSIP request and receives as parameters the P2PSIP message to send as well as a reference to the object to be notified when the reply to the sent message is received.
- *public void sendP2PSIPreply(P2PSIPMessage msg)* - This method is used for sending a reply message to a received P2PSIP request. As parameters, it only has to send the P2PSIP message in response to the request.

Both methods use the *ConvertToSIP* class to carry out the conversion of the *P2PSIPMessage* Java object (containing the information about the P2PSIP message to send) into an object recognized by the SIP layer in use.

When a SIP message is received by this layer, its headers are parsed in order to check whether or not it is a P2PSIP message and to determine if it is a request or a reply. If the received message is a valid P2PSIP request, it is converted to a P2PSIP object and passed to the upper layer. If the received message is a P2PSIP response, it is also converted into an equivalent P2PSIP object and passed directly to the object that sent the original request. The state of all ongoing requests is maintained by this layer.

B. DHT Layer Implementation

The DHT Layer is responsible for the implementation of the functionalities of the DHT algorithms in use. The major class in this layer is *Peer*, and it uses classes *ResourceMap* and *DHT* to maintain resources and the overlay. The two DHT algorithms implemented (Chord [3] and EpiChord [4]) are subclasses of *Peer* that rewrite some of its methods. New algorithms can be added in this way.

1) Chord implementation: The implementation of the Chord algorithm was based on the draft specification [13] from the same authors of dSIP. S. Cyrano et al [5] developed an application that implements the dSIP protocol using Chord and Kademlia algorithms. The application source code is available online [5] and was partly used in the implementation of Chord. In our implementation, the class hierarchy as well as some of its components, such as the component responsible for creating the P2PSIP messages were reused. All the reused components suffered a significant amount of change, in order to create a higher level of abstraction between the DHT and SIP layer. In addition to this, the SIP stack used in [5] (MJSIP), was replaced by the Jain-SIP in this work.

2) EpiChord implementation: For the EpiChord implementation, we also followed the draft specification [13] that describes how the chord should be implemented in dSIP. One of the main points of the EpiChord implementation is the cache management because it is completely different from the management of the Chord finger table. The EpiChord cache is filled with information obtained from received messages. It is important to detect and remove dead entries which failed a certain number of times or whose lifetime exceeds a preset value. The management of the cache is very important for the proper functioning of the algorithm being as the location of resources is based on the information contained in the cache. In addition to this, due to the fact that there is no maximum number of entries in the cache, the detection and removal of entries that have already expired is important.

Another aspect that must be taken into account in what concerns the implementation of this algorithm, is the manner in which resources are inserted and located in the overlay. While Chord uses a mechanism based on simple exchanging messages to insert or locate resources, in EpiChord, the process is slightly more complex. It uses a mechanism based on sending messages in parallel.

3) Message headers: Our implementation of Chord and EpiChord algorithms did not change any of the SIP message headers defined in dSIP. The fashion in which the *DHT-Link* header must be processed was merely re-defined. In both Chord and EpiChord implementations, *DHT-Link* header is used by a peer to send information related to his successor, predecessor, or elements of the routing table (finger table in Chord, and cache in EpiChord) to another peer. According to [13], the link parameter of the header must possess one the following values:

- Pn - information about the nth predecessor of the peer.
- Sn - information about the nth successor of the peer.
- Fn - information about the nth entry of the finger table.

The n must be a positive value and for links of the type S or P it indicates their depth. For example, if the link parameter has the value $P1$, it means that the *DHT-Link* header contains information about the peer's immediate predecessor and if the value is $S5$ it means that the *DHT-Link* header contains information about its fifth successor. If the value is $S0$ or $P0$, it means that the header contains information about the peer itself. For type F, the n references the index of the entry in the finger table of the peer.

Figure 4 presents an example of a *DHT-Link* header used in the implementation of Chord in dSIP. In this example, the header possesses information regarding the immediate successor ($S1$) of the peer sending the message. The relevant attributes of the peer such as its identifier (peer ID) and IP address are visible in this header.

```
DHT-Link: <sip:peer@192.0.2.1;peer-ID=671a65bf22>;
link=S1;expires=600
```

Fig. 4. Chord - DHT-Link header of dSIP protocol

The contents of the *DHT-Link* header in the implementation of the EpiChord algorithm is slightly different especially in the *link* and *expires* parameters. Since EpiChord uses a cache instead of a finger table, the value Fn that the *link* parameter can possess, no longer makes sense. Therefore, in the implementation of EpiChord, Fn was replaced by a new value C (no index), which indicates that the header contains information related to a chache entry.

Another difference in the EpiChord implementation has to do with the meaning of the parameter *expires* of the *DHT-Link* header. EpiChord does not use the *expires* parameter in its cache entries. It uses a parameter named *lifetime*, instead. It has a different meaning. To avoid creating a new header in which the only difference would be that the *expires* parameter would be called *lifetime*, it was decided that the same header from the Chord implementation be used. In this case, the *expires* parameter contains the value of the lifetime.

V. Tests and Results

The proposed architecture was fully implemented in Java and the resulting prototype was exhaustively tested in realistic environments. First emulation in a single Linux system was used and later on, a Linux cluster was used. An initial group of tests was conducted in order to simply validate the implementation. Previous results published by the authors of EpiChord, using simulation tools, were used as a reference to validate this implementation. In the second group of tests, the goal was to evaluate the benefits of a having a two layer hierarchy when compared to a flat overlay in the presence of nodes with limited resources.

Table I summarizes the parameters used in all tests. Table II shows the two metrics considered to analyze the results: the average lookup time in seconds required to locate a resource in the overlay and the average number of hops.

A. Testing scenario

In all tests, each peer (or client) in the P2P overlay performs intensive lookup operations for a set of pre-configured resources. The developed application was modified so that each peer carries out approximately two lookups per second.

Number of peers	32 peers in preliminary tests 200 peers in major tests
Lookup rate	two lookup operations per second for each peer
Stabilization time	60 seconds
Timeout	5 seconds
DHT Algorithms	Chord (iterative mode) EpiChord with 3 variants: (P=1;L=1), (P=3;L=3) and (P=5;L=3)
Overlay Configuration	Flat (with and without limited peers) Hierarchical (two levels, with limited clients)
Exec Environment	Ubuntu 11.04 with CORE on Intel, 1GB RAM CentOS Linux cluster nodes (4 CPUs, 8GB RAM)

TABLE I
Parameters and values used in testing scenarios

Metric	Description
Lookup Time	Average lookup time required to locate a resource in the overlay
Number of hops	Number of hops in the overlay needed to find a resource

TABLE II
Performance metrics

The list of resources that each peer should try to locate is predefined (previously created and stored in a configuration file), forcing the same lookup sequence in each test for each peer. Thus, the results obtained from different tests can be compared.

Some peers were named as bootstrap peers. All initialized peers receive a given list of bootstrap peers to which they should try to connect to in order to join the overlay. When initializing, each peer establishes a connection with the first peer on the list. The other peers are used if the selected peer does not respond. In each test, 6 peers were chosen to act as bootstrap peers. In order to prevent a bootstrap server from being overloaded during initialization (only because it is the first on the list), a Round Robin technique was used to rotate the list of bootstrap peers that each peer receives.

The booting process of peers and clients was made using a bash script which initializes the Java application of each peer or client with various parameters according to the test to be performed. First, the bootstrap peers are initialized and then the remaining peers. If the test uses clients, these are the last to be initialized. The JAVA application receives various parameters to configure itself such as for instance, the DHT algorithm to use. One of these parameters is used to set how long the application should wait before starting to locate a resource. Being as the tests are focused on lookup performance, the value of this parameter was set to the $90s$. Thus, the lookup tests only start when the overlay is stabilized and the results are more reliable.

B. Emulated execution environments

The first tests were conducted on a single Linux machine using the *Common Open Research Emulator* (CORE) [14]. The CORE [14] is a tool which emulates newtorks on one or more machines. Networks created in CORE can connect to other emulated networks and/or real networks.

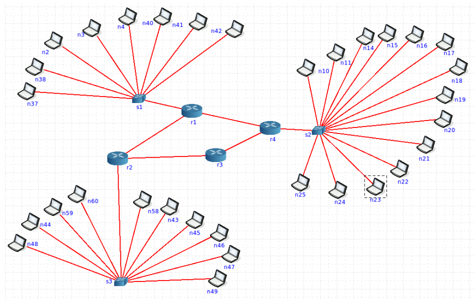


Fig. 5. Topology used to perform tests

The network topology created in CORE consists of 4 routers, 3 switches and 32 hosts (peers or clients), as shown in Figure 5. The topology was configured in order to replicate the testing conditions that were used by the authors of EpiChord. The objective was to validate the implementation by comparing the results obtained with the simulation results published in [4]. For this reason, physical links between hosts and the switches in the topology were set with a delay of $80ms$. This results in an average Round Trip Time (RTT) between peers of $0.16s$. This is the same value that was used in the tests conducted by the authors of EpiChord.

Due to the limitations of the machine on which CORE was executed, the maximum number of peers supported was 32 and therefore far from the minimal of 200 reported in [4]. It was therefore necessary to use a linux cluster (SeARCH [15]) to perform tests with a larger number of peers (200 peers at least). Using a restricted access account, we were unable to install CORE on the cluster and use it to create the network topology. The alternate solution found was to use multiple cluster nodes with multiple instances of the application in execution in each node. While in a real topology each peer has a different IP address, in this case, all instances of the application running on the same node share the same IP address. Therefore, in our tests, instead of varying the IP address of peers, each peer used a different port number.

Of the various performance metrics to be analyzed in the tests, the only one that could be affected by this solution is the *lookup time*. The parameter *number of hops* is not affected because it refers to the number of hops in the overlay and not in the physical network. In this execution scenario, connections between peers are often made on localhost. In this case, the delay in message delivery is virtually nil. To solve this problem, the developed application was modified to simulate a delay. When a message is received, instead of being processed immediately, the application makes a short waiting period in order to simulate the delay the message would experience within a real network. This simulated delay is a random value between two parameters, *rttMin* and *rttMax*, passed to the application.

Before testing with a greater number of peers, it was necessary to verify whether or not the results obtained originally in CORE with 32 peers were identical to the results obtained with 32 peers with the cluster using the solution described and with the same delays (between peers). The results in both tests were similar so we concluded that our solution was valid. The remaining tests were made in the cluster. The tests performed with 200 peers required the usage of two cluster nodes with 8GB of RAM and 4 CPUs each.

C. Chord and EpiChord configuration

The parameters of Chord and EpiChord algorithms used in the tests are the same that the authors of EpiChord used in their tests. However, some parameters could not be used because they were not mentioned in their tests. One such exampe is the number of entries in the Chord finger table. In [13] the recommended value is 16 for small networks and 32 for large ones. The chosen value is 32.

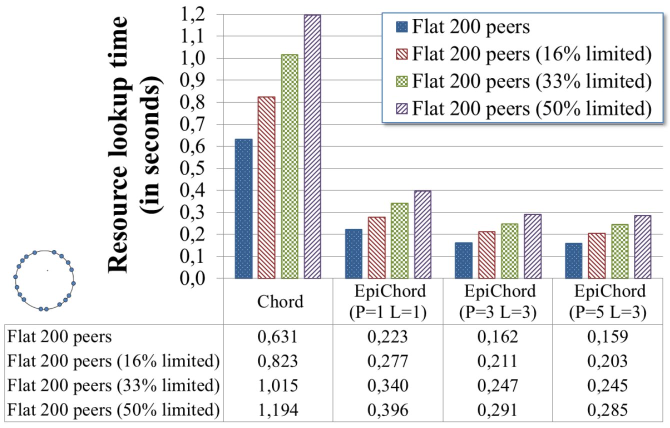
In the Chord algorithm, one minute ($60s$) was chosen as the stabilization period. This means that every minute each peer will check the state of their direct neighbors (predecessor and successor). In addition to the stabilization period, a period to check the finger table entries was also set. The value is also not specified in the tests made by the authors of EpiChord, so it was defined in the $70s$. The value in the range of values recommended in [13].

In the EpiChord implementation, a stabilization period of $60s$ and a maximum lifetime for the entries in cache of $120s$ was also used. These were the same values used by the authors of EpiChord in their tests. With regards to the level of parallelism in the location of resources, our tests were made with three different levels of parallelism. The following combinations were tested: $P = 1$ $L = 1$, $P = 3$ $L = 3$ and $P = 5$ $L = 3$, where P indicates the number of messages to be sent in parallel to begin the process of locating a resource or peer, and L is the number of contacts that each peer sends in its response messages.

Another parameter, common to both algorithms which was also defined by the authors of EpiChord in their tests, is the timeout value. The EpiChord authors used $0.5s$ as the value for this parameter in their tests. However, in our tests, this value produced a high number of lost messages. Therefore, $5s$ was used as timeout. The reason why the timeout value used by the authors of EpiChord did not work practically is probably due to the fact that this value has been used in simulation and not a in real environment. Even considering that the application was developed to support concurrency (multi-threading), EpiChord was implemented on top of another protocol (SIP), and the SIP stack itself defined $32s$ as an optimal timeout value. The values had to be adjusted. Therefore, in this work $5s$ was used as the timeout value.

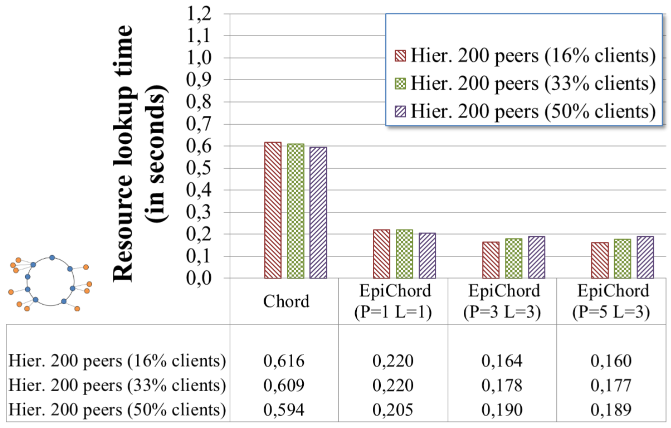
D. Flat P2P overlay

Figures 6a and 6c depict the results of the tests conducted with 200 peers. The first column, for each DHT algorithm, shows the results when all peers have similar resources (equally capable). By analyzing the results and comparing them with the results obtained by the authors of EpiChord in their simulation tests (See [4], Fig. 11 and 12 for 200 nodes),



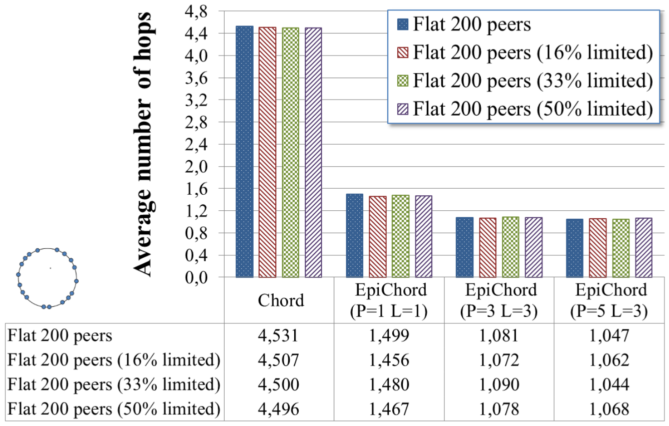
	Chord	EpiChord (P=1 L=1)	EpiChord (P=3 L=3)	EpiChord (P=5 L=3)
Flat 200 peers	0,631	0,223	0,162	0,159
Flat 200 peers (16% limited)	0,823	0,277	0,211	0,203
Flat 200 peers (33% limited)	1,015	0,340	0,247	0,245
Flat 200 peers (50% limited)	1,194	0,396	0,291	0,285

(a) Flat P2P overlay (lookup time)



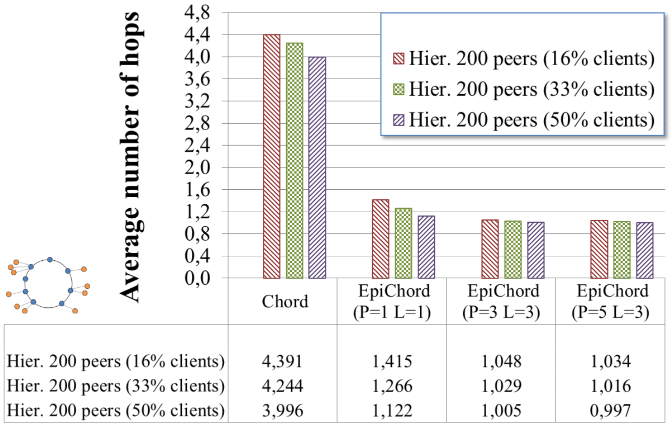
	Chord	EpiChord (P=1 L=1)	EpiChord (P=3 L=3)	EpiChord (P=5 L=3)
Hier. 200 peers (16% clients)	0,616	0,220	0,164	0,160
Hier. 200 peers (33% clients)	0,609	0,220	0,178	0,177
Hier. 200 peers (50% clients)	0,594	0,205	0,190	0,189

(b) Hierarchical P2P overlay (lookup time)



	Chord	EpiChord (P=1 L=1)	EpiChord (P=3 L=3)	EpiChord (P=5 L=3)
Flat 200 peers	4,531	1,499	1,081	1,047
Flat 200 peers (16% limited)	4,507	1,456	1,072	1,062
Flat 200 peers (33% limited)	4,500	1,480	1,090	1,044
Flat 200 peers (50% limited)	4,496	1,467	1,078	1,068

(c) Flat P2P overlay (hops)



	Chord	EpiChord (P=1 L=1)	EpiChord (P=3 L=3)	EpiChord (P=5 L=3)
Hier. 200 peers (16% clients)	4,391	1,415	1,048	1,034
Hier. 200 peers (33% clients)	4,244	1,266	1,029	1,016
Hier. 200 peers (50% clients)	3,996	1,122	1,005	0,997

(d) Hierarchical P2P overlay (hops)

Fig. 6. Intensive resource lookup on flat and hierarchical P2P overlay with 200 peers

it is possible to observe that they are very similar. Chord and EpiChord algorithms are therefore working as expected and were well integrated in the overwall architecture presented in Figure 3. Further DHT algorithms can be integrated in the future with little effort and little knowledge of the other modules.

However, the average number of hops in resource location of the chord algorithm is slightly higher than in our implementation. The reason for this may be due to the number of entries in the finger table. While finger tables with 32 entries were used in our tests, the tests made by the authors of EpiChord did not mention this value.

In addition to the verification of the correct operation of the implementations, these results also indicate that the use of EpiChord improves the performance of resource location. It improved about 72% of the lookup time as well as an average of 73% of the number of hops needed to find a resource.

E. Flat P2P overlay with limited peers

In order to verify if an overlay benefits from the existence of a hierarchy composed of peers and clients, where clients can be peers with lower capacities, tests were conducted where peers with connectivity limitations were introduced in the overlay. For a fair comparison of results, these tests were conducted under the exact same conditions except for the connectivity limitations introduced in some topology peers. For those selected peers, the connection delay was changed from $80ms$ to $200ms$. Thus, the average RTT of peers in the overlay varies between 0.16 and $0.40s$. In order to verify the impact of such peers in the overlay, different scenarios were created and more peers with limitations were introduced. This would permit the assessment of the impact on the performance of the overlay in resource location.

Figures 6a and 6c show also the results of the tests performed on the following scenarios:

- Scenario 1 - Connectivity limitations in 16% of peers
- Scenario 2 - Connectivity limitations in 33% of peers
- Scenario 3 - Connectivity limitations in 50% of peers

Analyzing the data in figure 6a, it is evident that the existence of peers with limited connectivity influence the performance of the overlay for resource location. As would be expected, scenario 3 presents the greatest increase in the lookup time because it possesses the largest number of peers with limitations (100 in 200). The average lookup time for resource location in this scenario increased between 84%

and 89% in Chord, and 77% to 83% in all tested variants of EpiChord. Analyzing the other two scenarios, it is also clear, as expected, that the increase in resource lookup time is directly connected to the increase in the number of peers with limitations. These results also show that sending messages in parallel with EpiChord, increases its immunity to problems caused by limited peers.

F. Hierarchical P2P overlay

In order to demonstrate the benefit of the existence of a hierarchy in a P2P overlay, we repeated the previous tests exchanging limited peers by clients. The tests were made with 200 nodes (between peers and clients), in the resulting two-level hierarchy, for the following scenarios:

- Scenario 1 - 16% of the nodes are clients (32 in 200)
- Scenario 2 - 33% of the nodes are clients (66 in 200)
- Scenario 3 - 50% of the nodes are clients (100 in 200)

The results obtained with this two-level hierarchy consisting of peers and clients are presented in Figures 6b and 6d. By analysing the average lookup times (6b) and comparing them with the values obtained from the previous test (6a), it is visible that the performance of the overlay in resource location benefits from the existence of a hierarchy. The results indicate that the exchange of the constrained peers by clients, keeping these limitations, allows the overlay to improve its performance. Analyzing the results of scenarios 1 and 3, we conclude that the overlay can improve performance from 33% to 100% in the Chord, from 26% to 93% with EpiChord without parallelism, and from 24% to 50% in EpiChord variants using parallelism.

VI. Conclusions and Future Work

In this article, a pure SIP framework capable of building hierachical P2P newtorks was presented. The current Java implementation includes two DHT algorithms, Chord and EpiChord, but was designed to easily plug new ones. The communication between peers is entirely done by the P2PSIP protocol dSIP. The developed implementation allows the creation of P2P overlays with one or two hierarchical levels. To support the two-level hierarchy, the dSIP protocol was extended. New headers were specified in order to allow communication between peers (belonging to the overlay) and clients (outside the overlay).

The results obtained from a flat P2P network are comparable with the results obtained by the authors of EpiChord using simulation. The results also indicate that increasing the number of peers with limitations in the overlay significantly degrades the overall performance of the overlay in resource location. The results obtained on a hierachical P2P network, with two levels, in which the peers with limitations become clients with the same limitations, show that the existence of a two-level hierarchy benefits the overlay performance. In this case, the performance of the overlay is identical to the performance achieved when the overlay has peers without limitations.

When comparing the results of the various tests made, with Chord and with EpiChord without parallelism (ie, parameters P and L with a value 1), it is visible that the use of a cache in EpiChord improves the performance of the overlay.

Our next step is the design and implementation of a mechanism that can dynamically promote clients to peers and peers to clients. The parameters taken into consideration for promotion and demotion must be carefully analyzed in order to prevent this mechanism from having a negative effect on the overlay performance. As a future work, we also intend to conduct additional tests with a significantly greater number of hosts (peers and clients). The goal is to reproduce, in a more realistic environment, results that are usually obtained by simulation. In such an environment the advantages and disadvantages of using SIP to create a P2P hierarchical overlay can be better analyzed.

Acknowledgment

This work is partially funded by FEDER Funds through the Programa Operacional Fatores de Competitividade – COMPETE and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-022674

References

[1] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "A SIP usage for RELOAD," *Internet Draft*, Jan. 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-sip-07

[2] D. Bryan, "dSIP: a P2P approach to SIP registration and resource location draft standard," http://tools.ietf.org/html/draft-bryan-p2psip-dsip-00, Feb. 2007.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[4] B. Leong, B. Liskov, and E. D. Demaine, "Epichord: Parallelizing the chord lookup algorithm with reactive routing state management," *Computer Communications*, 2006.

[5] S. Cirani and L. Veltri, "A kademlia-based DHT for resource lookup in P2PSIP," http://tools.ietf.org/html/draft-cirani-p2psip-dsip-dhtkademlia-00, Oct. 2007.

[6] G. Camarillo, "Peer-to-Peers (P2P) architecture: Definition, taxonomies, examples, and applicability," RFC 5694 (Informational), Nov. 2009.

[7] 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 and Tutorials*, 2005.

[8] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," *Peer-to-Peer Systems*, pp. 53–65, 2002.

[9] H. Schulzrinne, M. Matuszewski, and S. Baset, "Peer-to-Peer protocol (P2PP)," http://tools.ietf.org/html/draft-baset-p2psip-p2pp-01, 2007.

[10] C. Jennings, S. Baset, H. Schulzrinne, B. Lowekamp, and E. Rescorla, "REsource LOcation and discovery (RELOAD) base protocol," http://tools.ietf.org/html/draft-ietf-p2psip-base-22, Jul. 2012.

[11] D. Bryan, B. Lowekamp, and C. Jennings, "Sosimple: A Serverless, Standards-based, P2P SIP Communication System," *AAA-IDEA*, 2005.

[12] M. Ranganathan, P. O'Doherty, and J. van Bemmel, "JAIN-SIP: Stack sip for java." [Online]. Available: http://jsip.java.net/

[13] D. Bryan and M. Zangrilli, "A chord-based DHT for resource lookup in P2PSIP," http://tools.ietf.org/html/draft-zangrilli-p2psip-dsip-dhtchord-00, Feb. 2007.

[14] J. Ahrenholz, "Comparison of CORE network emulation platforms," in *IEEE Military Communications Conference*, 2010, pp. 864–869.

[15] SeARCH, "SeARCH: Services and advanced research computing," Cluster do Departamento de Informatica, Universidade do Minho. [Online]. Available: http://search.di.uminho.pt