Resource Location in P2P Systems

João Pedro Fernandes Alveirinho
joao.alveirinho@ist.utl.pt

Instituto Superior Técnico

(Advisor: Professor Luís Rodrigues)

Abstract. Peer-to-peer (P2P) systems have emerged as a potential technology to build very-large distributed data sharing systems. A key problem in these systems is the location of resources. This report makes a survey of the main strategies to implement resource location in P2P systems and identifies some possible lines of research to improve the current state of the art.

1 Introduction

Since the appearance of Napster[1] in 1999, P2P systems have been the subject to intensive research and development efforts, both in academia and industry. Peer-to-peer file sharing systems such as Gnutella[2], eMule[3], Kazaa[4] and more recently BitTorrent[5] have had tremendous success. Unfortunately, these systems have been mainly used to illegally distribute copyrighted material. Fortunately, examples of legitimate uses of this technology also exist and can be found in[6–8]. From the technological point of view, the potential of the technology to build extremely large-scale shared repositories makes it a very interesting research topic.

One of the main challenges in P2P systems is how to efficiently support resource location. Due to scalability and dependability issues, centralized solutions are not adequate. On the other hand, exhaustive search on all peers is also a non-scalable solution. Therefore, it comes as no surprise that resource location algorithms have been intensely studied, and many different solutions have been proposed in the literature.

As it will be surveyed in the report, there are two main types of P2P systems: structured and unstructured systems. Typically, structured P2P systems implement a *distributed-hash-table* (DHT), such as Chord[9], CAN[10] and Pastry[11]. These systems are highly optimized to implement exact-match queries, but provide poor support for more complex inexact queries. In addition, structured systems may be expensive to maintain in highly dynamic environments. An alternative to this approach are unstructured P2P systems, that have low maintenance cost but poor support for query operations. The most basic approach to implement resource location in unstructured systems is the use of limited flooding, an extremely expensive solution that can however, easily support complex queries. In between these two extreme solutions, many algorithms

have been proposed, including content-based[12], probabilistic[13], and index-based[14] resource location algorithms.

This report makes a survey of the main techniques that have been proposed in the literature to support resource location in both structured and unstructured P2P systems. From the analysis of these solutions, the report makes the case for an hybrid solution that combines both structured and unstructured support in order to improve the current state of the art.

The remaining of the report is organized as follows. Section 2 describes the goals and motivation for this work. Sections 3 and 4 provide the context, by surveying previous research, covering P2P overlays and resource location techniques respectively. Section 5 presents the proposed architecture. Section 6 describes how we plan to evaluate the proposed techniques. A schedule of future activities is provided in Section 7. Finally, Section 8 concludes the report.

2 Goals

This work addresses the problem of resource location in large-scale P2P systems. More precisely:

> *Goals:* This works aims at analyzing, designing, and evaluating algorithms to efficiently support complex queries in large-scale overlay networks.

Our approach to tackle this problem departs from the observation that structured and unstructured overlays both have advantages and disadvantages to support resource location. On the one hand, structured solutions provide fast and efficient exact-match search but lack flexibility to efficiently support complex or inexact queries. On the other hand, systems based on unstructured overlay networks support complex queries but are usually inefficient as they typically resort to flooding mechanisms that can be extremely CPU and bandwidth consuming. Therefore, we aim to explore new solutions that combine the usage of systems based on both unstructured and structured overlay networks in order to achieve a more efficient and flexible solution for locating resources in a P2P environment. In summary:

> *Expected results:* This work will: i) design and implement overlay topologies that simplify the implementation of resource location algorithms; ii) design and implement search algorithms that leverage on the previous topologies; iii) provide an evaluation of these algorithms based on simulations.

Depending on time constraints, we will also try to perform some experiments on the PlanetLab[15] platform.

3 P2P Overlay Networks

As the name implies, in a P2P system all nodes cooperate in a similar manner to achieve a common goal. In this report we focus on P2P systems that support distributed content sharing, however, it should be noticed that P2P systems can be used for other purposes, such as content distribution[6], cycle sharing[8], among others.

In a large scale P2P system, it is often undesirable or even impossible for each node to know and cooperate directly with each and every other node in the system. Instead, each peer only knows a small subset of all the system participants. The peering relations among nodes form a network; since this network is constructed on top of a physical (usually IP-based) network, it is called an *overlay* network, as illustrated in Fig. 1.

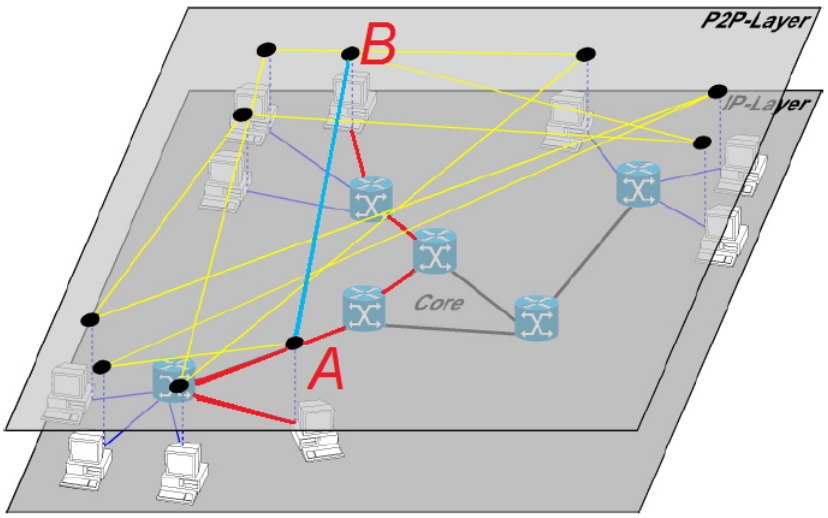

Fig. 1. Overlay Network

An overlay network, as any other network, can be modeled as a graph. There are some properties of the graph that are relevant for the operation of the overlay network. We list some of these properties below.

The *node degree* is the number of edges that connect the node. The graph is said to be *regular* if all nodes have the same degree. Typically, this is not the case in most overlay networks and, in fact, in some unstructured overlays, the node degree can vary substantially from node to node.

Another property is the *network diameter*, which is the number of edges that form the longest of all the shortest paths between any 2 nodes in the network. For instance, in Fig. 1 the underlying network diameter is 5.

Another property is *connectivity*. A network overlay is connected if there is a path that allows every node to reach every other node. If the overlay is connected nodes can rely on the overlay to communicate, and no peer is isolated from the rest of the system.

A property that affects both the network diameter and the connectivity of the network, is the *clustering coefficient*. The clustering coefficient of a node N is defined as:

$$C(N) = \frac{e(N)}{deg(N) * (deg(N) - 1)/2} \tag{1}$$

Where *deg(N)* is the node degree of N and *e(N)* is the number of links between 2 nodes that are also connected to N, i.e., N's neighbours.

A measure of the efficiency of routing in the overlay network is the *path stretch*: the ratio between the number of edges, of a certain path, in the underlying network and the overlay network. For instance, in Fig. 1 the stretch of the path between A and B is 5/1 since, the distance between A and B in the underlying network is 5 edges and the distance in the overlay level is 1 edge. It is worth noting that, when a path is used for communication, each edge crossed by a message is usually called a *hop*.

Finally, it should be noticed that overlays are not static, as nodes leave, crash and join the network. It is also possible that the overlay management protocols cause nodes to change their peers (neighbours), changing the overlay topology. A sequence of multiple join, leave, and crash events at a fast pace is a phenomena that often affect overlay networks that has been dubbed *churn*.

3.1 Unstructured and Structured Overlays

Overlay networks may be classified into two categories: *structured* and *unstructured*.

Unstructured Overlays In unstructured overlay networks, few or no constraints are imposed on the network topology, which means that the neighbours of each node may be chosen at random or emerge from the way the user interacts with the application. This makes these networks simpler to build and to maintain. For instance, in order to maintain overlay connectivity it is important for each node to maintain a minimum number of links to other nodes in the network (neighbours). When one of the neighbours fails or becomes disconnected, it should be replaced by another neighbour. Since unstructured networks require few constraints to be preserved, it becomes easier to find a suitable replacement.

Even if little effort is put in maintaining an unstructured network, it has been observed that many of these networks have (by emergence or by construction) *small world* properties, i.e., networks with a small average path length and large clustering coefficient[16]. These properties can be exploited in benefit of the operations that execute on top of the overlay. Furthermore, these networks are also naturally redundant and therefore more resilient to node failures and dynamic environments (such as churn).

Despite the characteristics listed above, unstructured networks pose challenges to the support of efficient resource location, because there is no correlation between the topology of the network and the location of the resources. The most simple manner to implement resource location on unstructured networks is through the use of flooding. However, flooding is very expensive and may cause network congestion due to the large number of redundant messages that it may generate. Therefore, flooding is not a scalable solution, unless optimizations are employed[17], as we will present later in this report.

Structured Overlays Structured overlay networks usually implement a distributed hash table (DHT). These overay networks, by construction, impose strong constraints on which nodes may be neighbours. Typically, these neighbours are determined by the node's *unique identifier* (*uid*). Objects stored in a DHT also own a unique identifier (commonly in the same identifier space as the nodes), which determines the node responsible for storing that object. DHTs support uid-based routing, tipically in a number of hops logarithmic with the system's size. This functionally can trivially be used to support exact-match queries. However, structured networks also have their own drawbacks. To start with, decomposing a complex query in a set of exact queries is often non-trivial or even impossible. Furthermore, in order to retain their structure, DHTs have significative associated maintenance costs. Structured overlays are also less resilient to identity forging attacks, for instance, sybil attacks[18], and to churn.

3.2 Flat and Hierachical Overlays

Overlay networks can also be classified into *flat* and *hierarchical* overlays. Hierarchical overlays usually consist of, as described in [19], "two-tier overlays whereby the peers are organized into disjoint groups". The overlay routing to the target group is done using an inter-group overlay and then, an intra-group overlay is used to route to the target peer.

Hierarchical unstructured networks are often built considering that nodes have heterogeneous capacity and stability, and can be classified into two categories: *regular* peers of lower capacity (and/or more volatile) and *super-peers* of higher capacity (and/or more stable). Super-peers coordinate a group of regular peers and coordinate with other super-peers to form an inter-group overlay.

In hierarchical DHTs each hierarchical group or set forms its own overlay[20] and, together they form a hierarchical overlay. Hierarchical DHT overlays offer several important advantages over flat DHT-based P2P overlays[19], namely:

- Reduce the average number of hops in a lookup query. Fewer hops per query implies less communication overhead. Also, if the higher-level overlay topology consists of stable superpeers the network itself will become more stable.
- Reduce the query latency when the peers in the same group are topologically close. In addition, the stability and the high capacity of the higher-level superpeers can also help to cut down the query delay.

– Facilitate large-scale deployment by providing administrative autonomy and transparency, while enabling each participating group to choose its own overlay protocol. Intra-group overlay routing is totally transparent to the higher-level hierarchy. If there are any changes to the intragroup routing and lookup query algorithms, the change is transparent to other groups and higher-level hierarchy. That is, any churn events within a group are local to the group, and routing tables outside the group are not affected.

3.3 Some Important Features

We now discuss a number of important features that any P2P system should own, regardless of the approach used to its construction. These features are: load balance, low maintenance overhead, scalability, and fault-tolerance.

3.3.1 Load balance The P2P system operation should balance the communication and processing overhead among all nodes of the network. For storage, each node should be responsible for an equivalent fraction of the objects being stored in the network. When supporting resource location, it is important that all the nodes receive an equivalent fraction of the queries being made.

The hash function used in structured P2P system may help to balance the load, but this is often insufficient, and additional load-balancing measures need to be considered. Some examples of load balancing algorithms/solutions are:

– The use of multiple hash functions to balance the storage of objects in a structured P2P system such as Chord[9] has been suggested in [21]. The basic idea is that each object may have several identifiers, each one generated by a different hash function and, therefore, increasing the number of nodes (at most as many as the number of hash functions used) where it can be stored. The object is stored in the node with the lowest load. Other nodes that could also be responsible for the given object may also store pointers to the object's location.
– In [22], the notion of virtual servers is explored to design load-balancing algorithms. The idea is that each node can have multiple identifiers and join a DHT in different logical locations. As a result, each node becomes responsible for noncontiguous intervals of the identifier space. To achieve load balance, identifiers may be migrated from one node to another.
– In [23] a new and very simple approach for balancing stored data between peers in a fashion inspired by the dissipation of heat energy in materials is presented. During thermal dissipation, a material warmer than its environment delivers energy to the surrounding area. This process continues until a balanced distribution of energy is reached in the overall system. Assuming a DHT architecture in which each interval of identifiers is stored at several nodes, 3 methods to balance the storage load are presented. Assuming f as the minimum number of nodes assigned to a specific DHT region (or interval), if more than f nodes are assigned to a specific interval, one or more of

them may be moved to a different interval. If $2f$ different nodes in the same interval are overloaded, then the respective interval can be split in two so that a node only has to manage half of the objects. Finally, nodes in low data load regions can be moved to and/or merged with overloaded regions reducing the load imposed by storing data.
- Overnesia[24] is an unstructured overlay that creates and maintains *nesos*, which are clusters of peers used for load-balancing and fault tolerance. The load of queries directed to each cluster is distributed among the members of the cluster.

3.3.2 Low Maintenance Overhead Another important aspect of any P2P solution is the overhead required to maintain its operation. This overhead tends to be higher in structured overlay networks, since their join and leave algorithms are more complex. It is also important to consider the relationship between churn and the bandwidth consumption at a peer for overlay maintenance traffic. There are two strategies to perform maintenance in any overlay:

- *Reactive Maintenance*: In a reactive approach, maintenance is only performed in response to some external event that affects the overlay (e.g. a node joining or leaving). For instance, in a DHT-based network ,a peer handles the failure or departure of an existing neighbour (or the new joining peer added to its neighbour table) by sending a copy of its new neighbour set to other peers in the system. To save bandwidth, a peer can send only differences from the last sent information.
- *Opportunistic or Cyclic Maintenance*: In this type of approach maintenance is performed periodically and usually involves the exchange of information of each peer with one or more neighbours. This process takes place independently of the peer detecting changes in its neighbour set.

3.3.3 Scalability Scalability is the capacity of a system to maintain or gracefully degrade its performance as it grows (in number of users, number of objects stored, etc). Early unstructured file-sharing systems such as Gnutella[2] suffered from scalability issues, mostly due to the need of flooding the network when performing queries. These scalability issues in unstructured networks, led some to propose DHT solutions to the wide-area file search problem. There are however several proposals to improve scalability in unstructured networks such as GIA[25].

3.3.4 Fault-tolerance This is an important feature of any large-scale P2P system, given that, as the number of members increases, the probability of node or link failures occurring also increases. It is therefore crucial for P2P systems to be able to sustain these type of faults. Otherwise, the whole system operation may be in jeopardy.

3.4 An Example of a Centralized P2P System

Napster The first prominent and popular P2P file-sharing system was Napster[1], which was solely dedicated to sharing music files. Napster used a centralized server-based service model, illustrated in Fig. 2[1], where the central server was used for indexing functions and to bootstrap the entire system. In Napster this centralized server was also responsible for executing the queries each node required. This design is simple but suffers from various problems such as:

Fig. 2. Centralized Server Model

- The central server represents a single point of failure, as the system is unable to operate without it.
- The central server is a bottleneck as it has to sustain all the query load in the system.
- Given that peers cannot cooperate without first contacting the central server, the network resources of the server are also a bottleneck.

3.5 Examples of Unstructured Overlay Networks

3.5.1 Gnutella Gnutella[2] has been proposed in 2000 as a fully decentralized P2P system based on an unstructured overlay network. Queries are supported using flooding techniques. In the initial Gnutella version (v0.4), each node would run a Gnutella client software and, on startup, it would have to find at least one other node that was already a part of the Gnutella network. Different methods can be employed this operation, including a pre-existing address list of possibly working nodes shipped with the software and web caches of known nodes (called Gnutella Web Caches). Once connected to a contact node, the client would request from the contact node a list of addresses of other nodes in the network. The client would then try to connect to those nodes, as well as to other nodes provided by the new neighbours, until it reached a certain quota. When a user wanted to perform a search, the client software would send the request to each

[1] Fig. 2 taken from http://en.wikipedia.org/wiki/Peer-to-peer

actively connected node. Historically (version 0.4 of the protocol), the number of actively connected nodes for a client was quite small (around 5), so each node that received the request, would then forward it to all its actively connected nodes, and they in turn forwarded the request, and so on, until the packet reached a predetermined number of "hops" from the sender (maximum 7). This number of "hops" is commonly known as TTL (Time-to-live). Unfortunately, this approach has some scalability issues, as nodes can easily become overloaded by simultaneous queries performed by different nodes.

Later in 2001 a new version of Gnutella was released (version 0.6), that addressed these scalability problems. This new version introduced the notion of super-peers, illustrated in Fig. 3[2], to which regular peers registered with the goal of reducing the signaling traffic. Flooding was now restricted to the super-peer level. In this system new super-peers are elected when: i) a super-peer leaves the network; ii) a super-peer has too many regular peers connected to it (denoted leaf-nodes); iii) a super-peer has too few leaf-nodes. This election was based on an estimate of the capacity of the peer, in terms of CPU, storage, bandwidth, and availability (uptime). Gnutella is therefore an example of a hierarchical P2P system.

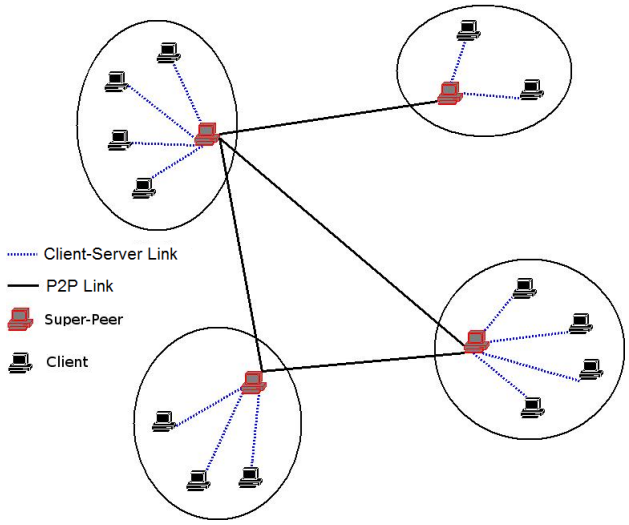

Fig. 3. Super-Peer Architecture

3.5.2 HyParView HyParView, which stands for Hybrid Partial View, is a gossip-based membership protocol[26] that builds and maintains an unstructured

[2] Fig. 3 adapted from http://schuler.developpez.com/articles/p2p/images/super-peer.jpg

overlay network. The protocol is characterized by each node maintaining two distinct views: a small and symmetric active view and a larger passive view.

The active views define an overlay that is used for cooperation among peers (the use of HyParView is illustrated with message dissemination applications). Links in this overlay are symmetric, which means that if node q is in the active view of node p then node p is also in the active view of node q. When a node receives a message for the first time, it broadcasts the message to all nodes of its active view (except, obviously, to the node that has sent the message). A reactive strategy is used to maintain the active view. Nodes can be added to the active view when they join the system. Also, nodes are removed from the active view when they fail.

On the other hand, the passive view is not used to support node communication. Instead, the goal of the passive view is to maintain a list of nodes that can be used to replace failed members of the active view. The passive view is maintained using a cyclic strategy. Periodically, each node performs a shuffle operation with one of its neighbours in order to update its passive view. In this shuffle operation, the node provides to its neighbour a sample of its partial views and, symmetrically, collects a sample of its neighbour's partial views. In fact, in this operation the identifiers that are exchanged belong not only to the passive view, but also to the active view. This increases the probability of having nodes that are active in the passive views and ensures that failed nodes are eventually expunged from all passive views.

This approach offers a strong resilience to node failures, even in the presence of extremely large numbers of crashes in the system. High resiliency to node failures is important to face unintentional (for instance, natural disasters) or intentional (for instance, software worms and virus) events that take down a significant portion of nodes in the system.

3.5.3 X-Bot X-Bot[27] is a protocol to bias the topology of an overlay according to some target efficiency criteria, for instance, to better match the topology of the underlying network, i.e., to reduce the average path stretch (it can be used to bias the topology for different criteria though). Based on HyParView, X-BOT relies on the combined usage of two distinct partial views; the goal of the protocol is to reduce the average link cost of the overlay network defined by the active views. For that purpose, X-BOT actively bias the neighbours in the active view using random peers extracted from the larger passive view. Moreover, the cyclic strategy used to maintain the passive view ensures that its contents are periodically updated and therefore, gives access to different potential neighbours over time to each node. Unlike HyParView, that strives to ensure the stability of the overlay, X-BOT relaxes stability to be able to continuously improve the overlay. This allows the topology of the unstructured overlay to self adapt to better match the requirements of the application executed on top of it. Periodically, each node starts an optimization round in which it attempts to switch one member of its active view for one (better) neighbour of its passive view. In the optimization protocol, a node uses its local Oracle to obtain an estimate of the

link cost to some random selected peers of its passive view. Examples of oracles are:

- *Latency Oracle:* This Oracle operates by measuring round trip times (RTT) to peers. This can be performed by exchanging probe messages with Oracles located at other nodes. The Oracle must be aware of the peers which are known at the local host, and it slowly measures the RTT for each know node (this value can be directly used as the cost value).
- *Internet Service Provider Oracle:* In a setting where exchanging messages across different ISPs has an increased monetary cost, it might be useful to keep as many neighbours as possible that share the same ISP. To this end, a simple oracle can be built by maintaining information concerning the local ISP and a table of costs for each known ISP. When the Oracle becomes aware of a new peer, it simply exchanges local ISP information with the remote Oracle and asserts the cost for the link using the local cost table.
- *IP-based Oracle:* These Oracles are able to calculate neighbour proximity values, which can be used as cost, using IP aggregation information (for instance, using a match of common IP prefixes to calculate a measure of proximity between two peers).

3.6 Examples of Structured Overlay Networks

3.6.1 Chord The first four DHT's (CAN[10], Chord[9], Pastry[11], and Tapestry[28]) were introduced at about the same time in 2001. Since then, this area of research has been quite active. Chord peers are organized in a flat circular (Fig. 4[3]) overlay network. Each node in chord, as in other DHT solutions, has its own identifier and so does each data item stored in the network. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Each key/data item pair should be stored at the node in the network that has the lowest ID which is still equal or bigger than the item's ID. Chord addresses difficult problems such as:

- Load balance: Chord acts as a distributed hash function, spreading keys evenly over the nodes, providing a degree of natural load balance.
- Decentralization: Chord is fully distributed and no node is more important than any other. This improves robustness and makes Chord appropriate for loosely-organized peer-to-peer applications.
- Scalability: The cost of a Chord lookup grows as the log of the number of nodes, so even very large systems are feasible.
- Availability: Chord automatically adjusts its internal tables to reflect newly joined nodes as well as node failures, ensuring that, barring major failures in

[3] Fig. 4 taken from [9]

the underlying network, the node responsible for a key can always be found. This is true even if the system is in a continuous state of change.

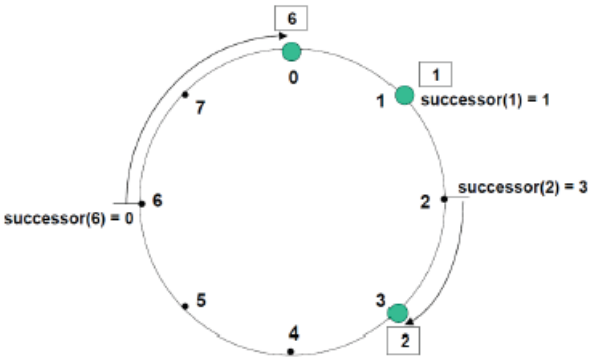

Fig. 4. Chord Architecture

In order to achieve efficient location of data items in the network, each node must mantain a reference to his successor in the ring and also a *finger-table* in which a set of routing entries are kept, to allow for larger hops in the overlay network. In this *finger-table* as can be seen in Fig. 5[4]:

- Each node mantains m entries (where m is the number of bits of the identifiers used).
- The i^{th} entry in the table at a node with an identifier n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle(s=successor(n+ 2^{i-1})).
- A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node.
- The first finger of n is the immediate successor of n in the circle.

Using the *finger-table*, Chord can efficiently route any query to its destination node. To achieve this, when a node n does not know the successor of a key k, that node will search its *finger-table* for a node j whose ID is closer (biggest predecessor of k) to k and ask j for the node it knows whose ID is closest to k. By repeating this process, n learns about nodes with IDs closer and closer to k. At each step of this process, the distance to the destination node is cut down by half, thus bringing n closer and closer until the successor of k is found. In fact, the number of steps of this process is logarithmic with the number of nodes in the system.

[4] Fig. 5 taken from [9]

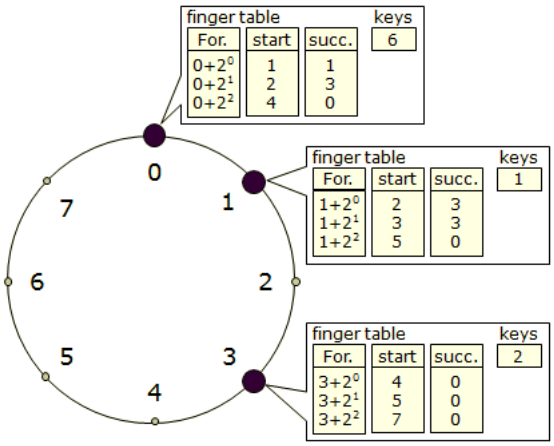

Fig. 5. Finger Tables

3.6.2 Kademlia Another example of DHT-based systems is Kademlia[29], which is a peer-to-peer $< key, value >$ storage and lookup system based on the XOR metric. Each node in Kademlia has 160-bit id chosen at random or by using an hash function such as SHA-1[30]. Keys are also 160-bit identifiers and, to publish and find $< key, value >$ pairs, Kademlia relies on a notion of distance between two identifiers (x and y) defined as the XOR (exclusive OR) of those two indentifiers (d(x,y) $= x \oplus y$). To route query messages, each node keeps contact information about other nodes. This means that, for each $0 < i < 160$, every node stores a list of $< IPaddress; UDPport; NodeID >$ triples for nodes of distance between 2^i and 2^{i+1} from itself. These lists, also known as k-buckets are kept sorted by last time seen (least recently seen node at the top) mainly because studies[31] have shown that the longest a node has been online, the more likely it is for that same node to remain online. Therefore, by keeping the oldest live contacts, k-buckets maximize the probability that the nodes they contain will remain online. For small values of i, k-buckets will generally be less populated or even empty (as few suitable nodes will exist). For larger values of i, the lists can grow up to size k, where k is a system-wide replication parameter. k should be chosen so that any given k nodes are very unlikely to fail or leave within an hour of each other, thus preventing the loss of the stored data.

Whenever a node receives a message from another node, it updates the correspondent k-bucket for the sender's node ID. This update operation follows a series of rules:

- If the sending node's ID already exists in the recipient's k-bucket then, the recipient moves it to the end of the list.
- If the node is not already in the appropriate k-bucket and the bucket has fewer than k entries, then the recipient inserts the new ID at the end of the list.

– If the correspondent k-bucket is full, then the recipient pings the k-bucket's least-recently seen node. If the least-recently seen node fails to respond, it is evicted from the k-bucket and the new sender is inserted at the end of the list. Otherwise, if the least-recently seen node responds, it is moved to the end of the list, and the new sender's contact is discarded.

This update strategy diminishes the need for cyclic maintenance to be performed since it keeps the k-buckets updated as a result of network traffic. For its operation, the Kademlia protocol relies on 4 RPC's:

– PING - This RPC probes a node to see if it is online.
– STORE (ID,value) - This RPC instructs a node to store a $< key, value >$ pair for later retrieval.
– FIND NODE(ID) - The recipient of a FIND NODE RPC should return $< IPaddress, UDPport, NodeID >$ triples for the k nodes it knows about closest to the target ID. These triples may come from a single k-bucket, or from multiple k-buckets, if the closest k-bucket is not full. Either way, the RPC recipient must always return k items (unless there are fewer than k nodes in all its k-buckets combined, in which case it returns every node it knows about).
– FIND VALUE(ID) - This RPC behaves like the FIND NODE RPC returning $< IPaddress, UDPport, NodeID >$ triples unless, the RPC recipient has received a STORE RPC for the key, in which case it just returns the stored value.

The most important procedure a Kademlia peer must perform is locating the k closest nodes to some given node ID. This procedure is called a *node lookup* and consists of a recursive algorithm in which:

– The lookup initiator starts by picking α nodes (where α is a system-wide concurrency parameter) from its closest non-empty k-bucket (or if that bucket has fewer than α entries, it just takes the α closest nodes it knows of).
– Then the initiator sends parallel, asynchronous FIND NODE RPC's to the α nodes it has chosen.
– The initiator then recursively sends the FIND NODE RPC to nodes it has learned about from previous RPC's until the initiator has queried, and collected responses, from the k closest nodes it has seen.

When $\alpha = 1$ the lookup algorithm resembles Chord's in terms of message cost and the latency of detecting failed nodes. However, Kademlia can route for lower latency because it has the flexibility of choosing any one of k nodes to forward a request to. Most operations are implemented using the *lookup procedure*. For instance, to store a $< key, value >$ pair, a participant locates the k closest nodes to the key and sends them STORE RPCs. Additionally, each node re-publishes the $< key, value >$ pairs that it has every hour in order to ensure persistence with very high probability.

3.6.3 Hieras The usage of Hierachical DHT's has been explored in works such as Hieras[20], Cyclone[32] and Canon[33]. Hieras is a multi-layer (Fig. 6[5]) DHT-based P2P routing algorithm. Like in other DHT's, all the peers in a Hieras system form a structured P2P overlay network. However, Hieras contains many other P2P overlay networks (P2P rings) in different layers inside the global P2P network. Each of these P2P rings contains a subset of all system peers. These rings are organized in such a way that the lower the layer of a ring, the smaller the average link latency between two peers inside it. In Hieras, the routing procedure starts in the lowest layer P2P ring in which the request originator is located and moves up until it eventually reaches the largest P2P ring. A large portion of the routing hops in Hieras are therefore taken in lower layer P2P rings, which have relatively smaller network link latencys. Therefore, an overall lower routing latency is achieved.

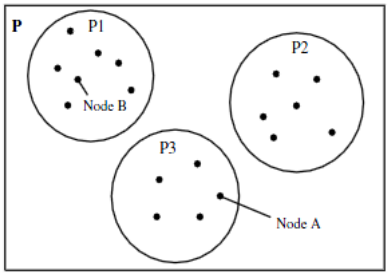

Fig. 6. Overview of a Two-Layer HIERAS System

In the Hieras's design, Chord was chosen as the underlying routing algorithm for its simplicity (it should however be easy to extend it to other DHT algorithms). A few changes have to be made to Chord's *finger tables* to comprise the hierarchical structure imposed by Hieras (Fig. 7[6]).

The original Chord *finger table* is used as the highest layer *finger table*. In addition, each node creates *m-1* (*m* is the hierarchy depth) other *finger tables* in lower layer P2P rings it belongs to. For a node to generate a lower layer finger table, only the peers within its corresponding P2P ring can be chosen and put into this *finger table*. Fig. 7 shows the *finger tables* of a node with the ID 121. This node's second layer P2P ring is "012". In the highest layer *finger table*, the successor nodes can be chosen from all system peers. For example, the layer-1 successor node in the range [122,123] is 124 and it belongs to the layer-2 P2P ring "001". Whilst in the second layer, finger table successor nodes can only be chosen from peers inside the same P2P ring as node 121. For instance, the

[5] Fig. 6 taken from [20]
[6] Fig. 7 taken from [20]

Start	Intervals	Layer-1 Successor	Layer-2 Successor
122	[122,123)	124 (“001”)	143 (“012”)
123	[123,125)	124 (“001”)	143 (“012”)
125	[125,129)	131 (“011”)	143 (“012”)
129	[129,137)	131 (“011”)	143 (“012”)
137	[137,153)	139 (“022”)	143 (“012”)
153	[153,185)	158 (“012”)	158 (“012”)
185	[185,249)	192 (“001”)	212 (“012”)
249	[249,121)	253 (“012”)	253 (“012”)

Fig. 7. Node 121 (012)’s Finger Tables in a Two-Layer HIERAS System

successor node in the range [122,123] is 143, which also belongs to layer-2 P2P ring ”012”.

4 Resource Location in P2P Systems

According to Webster’s Dictionary, to search is “to look into or over carefully or thoroughly in an effort to find or discover something”. In P2P Systems, in order to find the resources one is looking for, a search strategy must be applied. The choice of which search strategy to use depends not only on the type of network involved (structured or unstructured, hierarchical or non-hierarchical,...) but also on the results expected. Therefore, it is of extreme importance to understand which are the most common strategies and in which context they may be applied to achieve desired results.

4.1 Query Types

The selection of the query algorithm may depend on how the query is expressed. The simplest form of query is an exact match query, where objects that have a single attribute (for instance, a given name) are searched. However, it is often interesting to support richer forms of search, such as keywords, range-queries, or semantic queries as described below.

Exact Match Queries In exact-match queries, the object to be searched is specified by the value of a given attribute, for instance its name. DHTs are designed to support exact-match very effectively.

Keyword Queries Keyword queries are a generalization of exact match. Instead of characterizing the searched objects by a single attribute value, the objects to be searched are characterized by a logical expression that combined multiple attribute values (keywords) using *and* and *or* operators. For instance, one may search for a document tagged with string *s1* but not tagged with string *s2*.

Range Queries A range query aims at retrieving all records where some value is contained in some given interval. This type of query is commonly used in databases. An example of such type of query is: "*list all employees of a company with more than 2 and less than 6 years of experience*". Range queries are a challenging issue in the P2P search domain. They have been addressed by works such as Mercury[34].

Semantic Search A semantic search is a content-based, full-text search, whereby queries are expressed in natural language instead of simple keyword matches[35]. These types of queries aim at finding resources that are semantically similar to one described in the query itself. Semantic searches are even harder to support than range-queries: given a query, the system either has to search a large number of nodes or miss some relevant documents. Later in the document we will briefly overview pSearch[36], a system that supports this type of queries.

4.2 Characteristics and Performance Metrics

Peer-to-peer search algorithms may be optimized for different criteria. For instance, one might wish to get results faster at the cost of not being able to find all the possible results for a certain query. In this section, we list the most relevant characteristics and performance metrics of search algorithms.

Convergence A search algorithm is said to converge if, in each step executed by the algorithm, one becomes closer to finding the the desired object. This concept is clear in systems implementing exact-match queries on top of DHTs, where each hop in the lookup operation approximates the query to the target. In this case, the average number of steps for convergence is *log(n)* hops where n is the number of nodes in the network. In an unstructured overlay where exact-match is implemented by flooding, convergence is only ensured if a TTL with the diameter of the network is used (i.e., if the entire network is searched). Obviously, the cost of each step is quite different in both scenarios. Furthermore, when more complex queries are supported, the notion of convergence becomes blurred, given that it may be possible for the algorithm to return only a subset of the matching objects.

Recall Rate The query recall rate is defined as the ratio between the number of relevant documents retrieved during the processing of the query and the total number of relevant documents that existed in the system when the query was processed. For example, assume that when a query is placed there are in the network 60 objects that match the query but only 30 of those are returned; the recall rate for this query is *0.5*. Obviously, the higher the recall the better, although there may be a tradeoff between the recall rate and the cost of the query.

Message Cost Message cost can either be measured as the average or total number of messages necessary to execute queries, and as a result the amount of network traffic generated by such queries.

Latency Latency is a measure of how long it takes to obtain the response to the query. Latency can be measured in absolute time, which is actually the metric of relevance to the user, but that does not depends exclusively on the search algorithm but also on the properties of the IP network that supports the overlay. A more abstract manner to measure latency is to use the number of communication steps required to execute the query.

Precision Precision is defined as the fraction of the documents retrieved in a query that are relevant to the user's needs. Poor and inflexible query languages can lead to low levels of precision since users can experience difficulties when trying to describe the contents they are looking for. Therefore, one of the ways to improve a search mechanism's precision is to provide rich and flexible search languages and mechanisms so that its users can properly define the data they are looking for. Another approach to improve query precision is the usage of feedback mechanisms with which systems can learn from the user's input about query results. However, sometimes there might be a trade-off between the level of precision and the recall rate of a search algorithm. This means that, by relaxing the precision one can increase the algorithm's recall rate and vice-versa.

4.3 Query Dissemination Strategies

In order to execute queries in a P2P system, different query dissemination strategies may be applied. Each strategy has its own set of advantages and disadvantages, and the choice of using one or another is not always easy to make. The organization and structure of the peers in the overlay network is just one of the factors that affects this choice. This section aims at providing an overview of commonly used query dissemination strategies in P2P Systems.

Network Flooding As noted before, the basic and most straightforward approach to the search problem is network flooding. Flooding-based search was a popular approach in early unstructured P2P networks such as Gnutella[2]. In this strategy, the querying peer sends the query request to all or a subset of its neighbours. Then, each of these neighbours processes the query, returns the result if a match is found, and then forwards the query to its own neighbours. This procedure is repeated by each neighbour until a given TTL threshold is reached. This type of query dissemination mechanism, if the network is large (i.e. if each peer has a large amount of neighbours), generates a massive amount of network traffic per query. Gnutella used breadth-first search (BFS) and fixed TTL to limit the number of hops each query may take, in an attempt to reduce query bandwidth cost. Obviously, the downside of these cost mitigation strategies is the reduction of the query recall rate.

Iterative Deepening Iterative deepening is a variant of flooding that aims at reducing the number of nodes that are required to process the query[37] until a target number of responses is obtained. The basic idea is to initiate the query procedure by performing a limited flood with a small TTL; if the desired number

of responses is not achieved, a new flood with a larger TTL is initiated. This process is repeated until a maximum TTL is reached. Variants of this scheme are often called *expanded ring search*. An obvious limitation of this approach is that, when a new flood (with larger TTL) is initiated, all nodes involved in the previous flood need to re-execute the query. To prevent this behavior, when the query is re-sent, it is marked with the previous TTL, such that nodes already visited may simply forward the query without executing it. Alternatively, nodes at the border of the previous ring search may store the query for some time. In this case, the originator may just transmit a *resend* message, with the query identifier; when the resent reaches one of the border nodes, these "unfreeze" the corresponding query and forward it with the new TTL.

Random Walks This strategy is an alternative to flooding, that aims at avoiding the scalability issues posed by flooding on unstructured P2P systems[38]. Given a query, a random walk is essentially a blind search in which, at each step, the node that receives the query processes it and then forwards it to another single randomly chosen node. This process may go on until the query is satisfied and may be terminated in two ways: TTL and *checking*. TTL means that, similarly to solutions based on flooding, each random walk terminates after a certain number of hops in the overlay network, while *checking* means that a *walker* (i.e. the query message being forwarded in the network) periodically checks with the query originator before advancing to the next node. Random walks can effectively reduce the number of redundant messages, but at the cost of increasing search latency. To reduce this delay, one may use k-way random walks (k-*walkers*) where the querying node forwards the original query message to k randomly selected neighbours instead of only one. Since the number of nodes reached by k random walkers in h hops is the same as in one random walk over kh hops, a reduction of around k times in query delay can be expected.

Guided Searches Guided search is a search technique based on the construction, at each node, of indices that can "guide" the routing of the query[39]. Guided searches allow nodes to forward queries to neighbours that are more likely to have answers, rather than forward queries to randomly chosen neighbours or flood the network by forwarding the query to all neighbours. Routing indices indicate a promising *direction* toward the answers for queries. These distributed indices are small (i.e., compact summary) and provide hints on the probable *best* direction toward the resource one is looking for, rather than its actual location. Indices may be built incrementally, for instance based on the results of previous queries.

Probabilistic Search Probabilistic search is a name used to characterize a form of guided search based on incomplete information. These type of queries usually rely on the usage of data structures, such as *Bloom Filters*[40], or result caching mechanisms to determine when a certain peer or region of peers is likely to store a certain object or not. Because they rely on probability, some of these approaches may generate false positives and/or false negatives.

Similar Content Group-Based Search The basic idea behind this strategy is to organize P2P nodes into groups in which peers store similar content. This is usually applied on top of unstructured P2P systems such as Gnutella[2]. The intuition behind this approach is that nodes within a certain group tend to be relevant to the same queries. As a result, this type of search strategy will guide the queries to regions of nodes that are more likely to have answers to the queries, thus allowing to configure a flooding dissemination strategy with a smaller TTL value allowing these solutions to achieve a lower operation overhead. Several works such as [41] and [42] have addressed this type of strategy.

4.4 Examples of Systems Supporting Resource Location

This section illustrates how the techniques described previously have been used in different systems.

4.4.1 pSearch The fundamental challenge that the authors of pSearch[36] identified as being one of the causes for the complexity of P2P resource location solutions is that, with respect to semantics, documents are randomly distributed in the system. Therefore, given a query, a system has to either search a large amount of nodes or risk not being able to find relevant documents. To address this issue, the notion of semantic overlay is presented as "a logical network where contents are organized around their semantics, in such a way that the distance between two documents in the network is proportional to their dissimilarity in semantics". pSearch is a prototype P2P information retrieval system that works by representing documents as semantic vectors and organizing them in the network around their vector representations. In pSearch, to generate the semantic space, extensions to Vector Space Model (VSM)[43] and Latent Semantic Indexing (LSI)[44] are used, and CAN[10] is used to support the semantic overlay. CAN stands for content-addressable network. CAN organizes the logical space as a d-dimensional cartesian space and partitions it into zones. One or more nodes are responsible for each zone and every object key corresponds to a point in the space and is stored at the zone which contains that point. Locating an object in CAN is reduced to routing to the node that hosts the object. Routing from a source node to a destination node is equivalent to routing from one zone to another in the Cartesian space. Vector space model (VSM) represents documents and queries as *term vectors*. Each element of the vector represents the importance of a word (*term*) in the document or query. Two factors decide the importance of a term in a document: the frequency of the term in the document and the frequency of the term in other documents. If a term appears in a document with a high frequency, there is a good chance that term could be used to differentiate the document from others. However, if the term also appears in several other documents, the importance of that term is reduced. During a retrieval operation using VSM, the query vector is compared to document vectors and those closest to the query vector are considered to be similar and are returned. Latent semantic indexing was proposed to address synonymy, polysemy,

and noise problems in literal matching schemes such as VSM. For instance, although *car*, *vehicle* and *automobile* are different terms, they all reference the same or very similar objects. In VSM this would make no differnce but, LSI may be able to discover that they are related in semantics and therefore generate more reliable semantic vectors. The basic idea of pSearch is to use the semantic vector(generated by LSI) of a document as the key to store the document in the CAN.

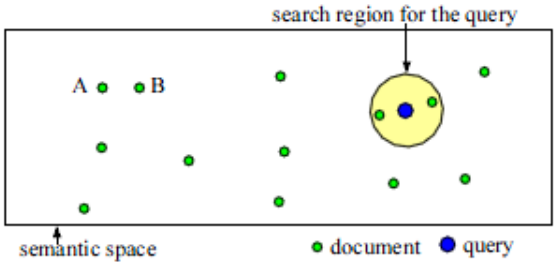

Fig. 8. Search in a Semantic Space

Fig. 8[7] shows how a semantic overlay can benefit searches. Each document is placed as a point in the (semantic) Cartesian space. Documents close in the semantic space have similar contents (e.g. documents A and B). Each query can also be positioned in this semantic space and to find documents relevant to that query it is only necessary to compare it against documents within a small region centered at the query, because the relevance of documents outside that region is relatively low. This results in an effectively smaller search space for the query.

To set-up the semantic overlay, an index for each document is stored in the CAN using the document's semantic vector as the key. Among other things, an index includes the semantic vector of a document and a reference (URL) to the document itself. The basic operation model in pSearch can be summarized in 4 steps:

- When receiving a new document (which can be submited by any node, inside or outside of the CAN) a, the Engine node, a node that is part of the CAN, generates its semantic vector Va using LSI and uses Va as the key to store the index in the CAN.
- When receiving a query q, the Engine node generates its semantic vector Vq and routes the query in the overlay using Vq as the key.
- When a query reaches its destination, it is flooded to nodes within a radius r, determined by the similarity threshold or the number of wanted documents specified by the user.
- All nodes that receive the query do a local search using LSI and report the references to the best matching documents back to the user.

[7] Fig. 8 taken from [36]

4.4.2 Cubit Cubit[45] takes a different approach on the issues discussed in psearch[36]. While the focus of pSearch is on finding documents with high semantic relevance to the search keys it is unable to match misspelled search keys to documents with correctly spelled keywords.To overcome this issues, cubit works by creating a keyword metric space that encompasses both the nodes and the objects in the system and where the distance between two points is a measure of the similarity between the strings that those points represent. The objective of the cubit system is to find the k closest data items for any given search key. This is achieved by creating a keyword metric space that captures the relative similarity of keywords, assigning portions of this space to nodes in the overlay and to resolve queries by routing them through this space. The focus of Cubit is on providing approximate keyword search for multimedia content with limited content description. Keywords are derived from the content's filename and information specific to the content type, such as the comment section of torrent files or the extended video information for YouTube video clips.

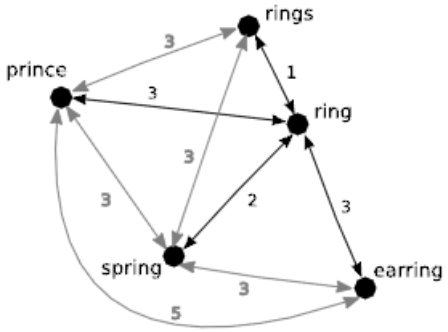

Fig. 9. The edit distance between keywords

An object stored in Cubit is characterized by one or more keywords. Cubit's approach to approximate matching relies on a notion of distance between keywords. Cubit mainly uses the most common notion of distance on strings, the *Levenshtein* distance, commonly known as the *edit* distance(Fig. 9[8]). It is equal to the minimum number of insertions, deletions, and substitutions needed to transform one string to another. However, search queries typically consist of more than one keyword (for instance, the title of a movie). Therefore, cubit matches queries using the phrase distance (i.e. the distance between two sets of keywords) which is used to calculate the distance between a query and an object.

In Cubit, nodes are distributed in the same space as keywords. This means that each node in Cubit is assigned a unique string ID chosen from the set of keywords associated with previously inserted objects in the system. This ID

[8] Fig. 9 taken from [45]

determines a node's position in the keyword space and each Cubit node is responsible for storing the set of keywords for which it is the closest node.

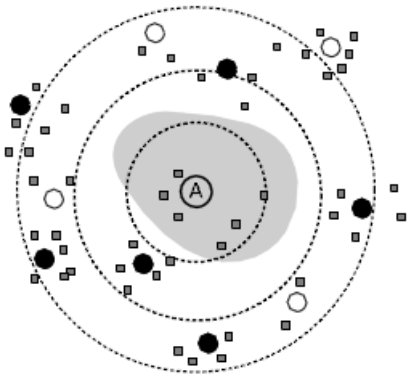

Fig. 10. Cubit concentric rings structure

In Cubit, a search operation is processed by a distributed protocol which navigates through nodes in the keyword space, gradually zooming in on a neighbourhood of a given (possibly misspelled) keyword, and thus locates nodes that store possible matches. To achieve this, Cubit creates and maintains a multi-resolution overlay network on nodes such in which each node has several peers at every distance from itself. Each Cubit node organizes its peers into a set of concentric rings. In each ring, a node retains a fixed number, k, of neighbours whose distance to the host lies within the ring boundaries. This ring structure enables a Cubit node to retain a relatively large number of pointers to other nodes in its surroundings, while also providing a sufficient number of pointers to far-away peers. This ring structure is depicted in Fig. 10[9] in which, the solid circles represent peers in node A's neighbourhood-set, the empty circles represent other nodes, and the squares represent object keywords in the system. The shaded region depicts the subspace that is closer to A than any other node. The master record for each keyword in the shaded region is stored at node A.

The way the Cubit search protocol operates is by iteratively collecting more and more information of the target region. In Fig. 11[10], x is the location of the search term in the keyword space, the solid circles are node A's peers, empty circles are additional nodes in the space, and the circle around x are all nodes within an edit-distance q of x. Node A first finds the $nmin = 2$ closest nodes to x from its neighbourhood-set, and requests their $nmin$ closest nodes. As a result, two new closer nodes are discovered and subsequently sent the same query. The protocol terminates when all nodes within the circle around x, or when the $nmin$

[9] Fig. 10 taken from [45]
[10] Fig. 11 taken from [45]

closest nodes have been discovered. These nodes are then queried for their objects closest to x.

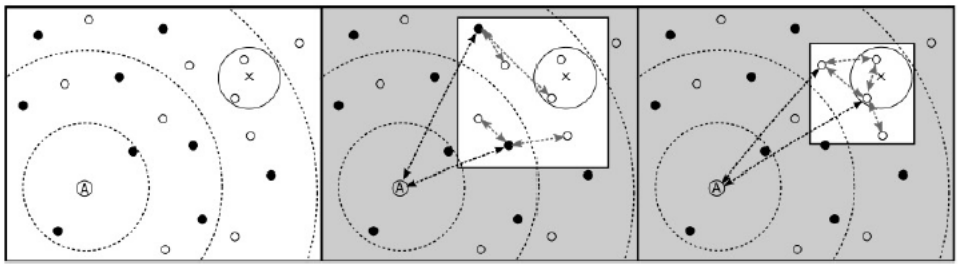

Fig. 11. The Cubit Search Protocol

4.4.3 Adaptive Probabilistic Search (APS) Search methods can be categorized as either *blind* or *informed/guided*. In *blind* searches, nodes do not store any information regarding file locations. In *informed/guided* approaches, nodes locally store metadata that assist in the search for the queried objects. *Blind* methods usually need to consume a lot of bandwidth to achieve high performance. On the other hand, *informed* methods use their indices to achieve similar quality results and to reduce traffic overhead. The problem with most *informed* methods is the maintenance cost of the indices after peers join/leave the network or update their collections. In most cases, these events trigger floods of update messages inflating network traffic.

In [46] a new search algorithm called Adaptive Probabilistic Search (APS) method is proposed. This algorithm achieves high performance at low cost. In APS, a node deploys k-walkers for object discovery, but the forwarding process is probabilistic instead of random. Peers direct these walkers using feedback from previous searches, while keeping information only about their neighbours.

The APS algorithm is defined for deployment over unstructured P2P networks and is based on some assumptions:

- Peers initiate searches for various objects that are distributed across the network according to a *replication distribution*, which dictates which objects are stored at each node.
- Popular objects get many more requests than unpopular ones.
- The search algorithms cannot dictate the placement and replication of objects in the system and are not allowed to alter the topology of the P2P overlay.
- A node is directly connected to its neighbours, and these are the only peers whose addresses the node knows about.

– Nodes can keep some soft state for each query they process. Each search is assigned an identifier, which, together with the soft state, enables peers to make the distinction between new and duplicate messages. Identifiers are also assigned to objects and nodes from a flat, non-hierarchical space.

In APS, each node keeps a local index per neighbour, consisting of an entry for each object it has requested, or forwarded a request for. The value of each entry reflects the relative probability of this node's neighbour to be chosen as the next hop in a future request for the specific object. In the forwarding process, a node chooses its next-hop neighbour according to the probabilities given by its index values and appends its identifier in the search message, keeping soft state about that search. If two walkers from the same request cross paths (i.e., a node receives a duplicate message due to a cycle), the second walker is assumed to have terminated with a failure and the duplicate message is discarded. Index values stored at peers may be updated using one of the following strategies:

– Optimistic Approach: In this approach, when a node chooses one or k (if the node is the query originator) peers to forward the request to, it proactively increases the relative probability of the peer(s) it picked, assuming the walker(s) will be successful
– Pessimistic Approach: In this approach, the node decreases the relative probability of the chosen peer(s), assuming the walker(s) will fail.

Upon walker termination, if the walker is successful, nothing is done in the optimistic approach but if the walker fails, index values relative to the requested object along the walker's path must be corrected. Therefore, using information included in the search message, the last node in the path sends an *update* message to the preceding node. This node, decreases its index value for the last node to reflect the failure and the update process continues along the reverse path towards the requester. If the pessimistic approach is used, the update procedure is analogous, and nodes increase the index values along the walker's path, but the update only takes place when a walker succeeds (instead of when a walker fails).

This method uses probabilistic walkers with a learning feature that incorporates knowledge from past and present searches to enhance future performance. The learning process adaptively directs the walkers to promising zones of the network. In fact, this method has an increased recall rate in comparison to the original *blind* k-walker[38]. APS also does not require message exchange on node arrivals or departures. If a node detects the arrival of a new neighbour, it will associate some initial index value to that neighbour and if a neighbour disconnects, each node that has that node as a neighbour, removes the relative entries and stops considering it in future queries.

4.4.4 SETS : Search Enhanced by Topic Segmentation SETS[41] is an architecture for efficient search in peer-to-peer networks, building upon ideas drawn from machine learning and social network theory. The key idea behind

SETS is to arrange sites (peers) in a network such that a search query probes only a small subset of sites where most of the matching documents reside. In particular, SETS partitions sites into topic segments such that sites with similar documents belong to the same segment. Each topic segment has a succinct description called the topic centroid. Sites are arranged in a segmented network that consists of two kinds of links. Short distance links connect sites within a segment. Long distance links connect pairs of sites from different segments.

When a search query is initiated, it is forwarded to other sites using a topic-driven routing protocol. First, topic centroids are used to select a small set of relevant topic segments. Next, the selected segments are probed in sequence. A probe to a particular segment proceeds in two steps: First, the query is routed along long distance links to reach a random site belonging to the target segment. Next, the short distance links are used to propagate the query to all/most/few sites within a segment. By applying this content-based search approach, SETS is able to reduce the number of nodes that are queried while mantaining a high recall rate since the nodes that are queried, are those with higher probability of holding the desired resources.

5 Proposed Architecture

To efficiently support complex queries in large-scale overlay networks, we propose an architecture that comprises 2 layers. The first of which will be composed by all the peers in the system, which will form an unstructured overlay network. The second layer will be composed by some selected peers, belonging to the first layer, which will in turn form a structured overlay network more precisely, a DHT. The way these two layers will interact in order to execute queries will be explained further ahead. For now though, a few observations must be made:

- We consider that each peer in the system has a set of resources (Music, Video, Documents, Processor,...) of its own, which it shares with the rest of the network.
- Each of these resources is characterized by a number of keyword tags (such as Music, Movie, TV Show, Game, Book, Application, Comics, ...).
- Under no circumstance is a peer required to store any resource other than the ones it chooses to share with the network.
- Peers may join or leave the network at any given time.

Unstructured Layer This layer will be composed by all the peers in the system, which will form an unstructured overlay. Peers in this overlay will organize themselves according to the type(s) of content they share. This will be done using X-Bot[27], with the type(s) of content shared by each peer as the biasing factor for the overlay topology. In particular, each peer will define the most relevant types of content (defined by the tags of each resource it shares) and try to find neighbours that share the same types of contents and keep them in its active views. For instance, if a peer defines *Audio* and *Games* as the most relevant types of contents it shares, the network topology will be biased (using X-Bot)

so that, that peer has some neighbours that have *Audio* as a relevant type of shared content, and also have neighbours whose relevant types of shared content include *Games*.

Structured Layer Some peers that belong to the unstructured layer, will also be selected to participate in a DHT overlay. This selection will be made according to peer capacity (e.g.: cpu and storage), bandwidth, availability (uptime), and also amount of shared content. Each peer that participates in this DHT, will register itself to one or more resource categories (keyword tags) which are amongst that peer's most relevant types of shared content. This means that for each resource category, there will be one or more peers who are representatives of that category in the DHT. This DHT will then be used by all the peers in the system to route their queries towards peers that share the type of content being searched for. For this, each peer in the unstructured layer will also mantain a small additional view containing some DHT participant peers.

Queries Whenever a peer wishes to perform a query in the system, that peer must define the category tags that the query is valid for (for instance, if the peer is searching for an audio book by, let's say, "Tolstoi", the tag *book* and *audio* should be used but for instance the tag *video* should not). Then, there are 2 possibilities for a peer p to execute this query:

- For each of the category tags, that the query is valid for, and that are amongst p's relevant types of shared content, p will use limited flooding in the unstructured layer. In this step, the query message will be forwarded to the subset of peers in p's active view that also share that type of content. That subset of peers will execute the query and also forward the query to their own subset of neighbours who also share that type of content. This process will continue until a previously defined value of TTL is achieved.
- For each category tag, that the query is valid for, that is NOT amongst p's relevant type of shared content, p will forward that query to the DHT using that category's hash as the destination, so that one of the peers in the DHT registered to that type of content receives the query. Then, that peer will disseminate the query using limited flooding in the same fashion as indicated in the previous item.

Main Advantages This solution should provide some relevant advantages:

- Query Flexibility: The only requirement is that the querying peer defines the resource categories for which that query is valid. The query itself can be of any type and the query language can be as rich and complex as desired. This presents an advantage when comparing to traditional DHT-based solutions which can only be used for *exact-match* queries.
- Directed Flooding: The usage of X-Bot with the type(s) of content shared by each peer as the biasing factor for the overlay topology, allows for the queries to be forwarded only to peers which share contents for which that query is

relevant. This, combined with the usage of a TTL value, should lower the amount of network traffic generated in each query, while still providing a high query recall rate (since the peers that receive the query are more likely to store relevant content).

6 Evaluation

The performance of the proposed solution will be evaluated both by simulation using Peersim[47] and, if time permits, via an experimental deployment in the testbed PlanetLab[15]. Simulations will be useful to test the proposed architecture under several scenarios in which the number of network participants will vary overtime. This will allow us to evaluate the solution's scalability as well as its resistance to churn events. In addition, it will also be crucial to analyze the level of maintenance overhead induced by the proposed solution.

On another note, the main performance metrics that will be analysed are:

- The Search Algorithm's Recall Rate.
- The Message Cost of Queries.
- The Search Algorithm's Precision.

These metrics will allow us to compare the search algorithm's performance against existing state-of-the-art solutions.

7 Schedule

Future work is scheduled as follows:

- January 9 - March 29: Detailed design and implementation of the proposed architecture, including preliminary tests.
- March 29 - May 2: Perform the complete experimental evaluation of the results.
- May 3- May 23, 2009: Write a paper describing the project.
- May 24 - June 13: Finish the writing of the dissertation.
- June 14, 2009: Deliver the MSc dissertation.

8 Conclusion

In this report we surveyed the most representative approaches to support resource location in large-scale P2P system. In order to provide the adequate context for this work, we have also introduced the main classes of P2P systems. We have observed that different classes of P2P systems have advantages and disadvantages for supporting resource location in an efficient manner. Finally, we have sketched a solution to implement resource location that is based on a combination of structured and unstructured overlays that we plan to develop in the future. We concluded the report with a description of the methodology to be applied for evaluating our solution.

Acknowledgments We are grateful to João Leitão, João Paiva and Mário Ferreira, for the fruitful discussions and comments during the preparation of this report. This work was partially supported by project "Redico" (PTDC/EIA/71752/2006).

References

1. : Napster `http://www.napster.com`.
2. : Gnutella `http://rfc-gnutella.sourceforge.net/`.
3. : emule `http://www.emule.com/`.
4. : Kazaa `http://www.kazaa.com/`.
5. : Bittorrent `http://bittorrent.org/beps/bep_0003.html`.
6. : Coral `http://www.coralcdn.org/`.
7. : Blizzarddownloader `http://www.wowwiki.com/Blizzard_Downloader`.
8. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Leboisky, M.: Seti@home-massively distributed computing for seti. Computing in Science and Engineering **3**(1) (Jan/Feb 2001) 78–83
9. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2001) 149–160
10. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01). Volume 31., New York, NY, USA, ACM (October 2001) 161–172
11. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (November 2001) 329–350
12. Totekar, C., Vani, M., Palavalli, S.: Search improvement in unstructured p2p network considering type of content. In: Networks, 2008. ICON 2008. 16th IEEE International Conference on. (Dec. 2008) 1–4
13. Rhea, S., Kubiatowicz, J.: Probabilistic location and routing. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 3. (2002) 1248–1257 vol.3
14. Zhang, R., Hu, Y.: Assisted peer-to-peer search with partial indexing. In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE. Volume 3. (March 2005) 1514–1525 vol. 3
15. : Planetlab `http://www.planet-lab.org/`.
16. Halim, F., Wu, Y., Yap, R.H.: Small world networks as (semi)-structured overlay networks. Self-Adaptive and Self-Organizing Systems Workshops, IEEE International Conference on **0** (2008) 214–218
17. Leitão, J., Pereira, J., Rodrigues, L.: On the structure of unstructured overlay networks. In: In "Fast Abstract", Supplement of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE (2008)
18. Danezis, G., Lesniewski-laas, C., Kaashoek, M.F., Anderson, R.: Sybil-resistant dht routing. In: In ESORICS, Springer (2005) 305–318

19. Buford, J., Yu, H., Lua, E.K.: P2P Networking and Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
20. Xu, Z., Min, R., Hu, Y.: Hieras: A dht based hierarchical p2p routing algorithm. Parallel Processing, International Conference on **0** (2003) 187
21. Byers, J., Considine, J., Mitzenmacher, M.: Simple load balancing for distributed hash tables. (2002) 80–87
22. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in structured p2p systems (2003)
23. Rieche, S., Petrak, L., Wehrle, K.: A thermal-dissipation-based approach for balancing data load in distributed hash tables. In: Local Computer Networks, 2004. 29th Annual IEEE International Conference on. (Nov. 2004) 15–23
24. Leitão, J., Rodrigues, L.: Overnesia: an overlay network for virtual super-peers (December 2008)
25. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2003) 407–418
26. Leitao, J., Pereira, J., Rodrigues, L.: Hyparview: A membership protocol for reliable gossip-based broadcast. In: DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2007) 419–429
27. Leitao, J.C.A., Marques, J.P.d.S.F.M., Pereira, J.O.R.N., Rodrigues, L.E.T.: X-bot: A protocol for resilient optimization of unstructured overlays. In: SRDS '09: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems, Washington, DC, USA, IEEE Computer Society (2009) 236–245
28. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **22** (2004) 41–53
29. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. (2002) 53–65
30. : Sha-1 `http://www.w3.org/PICS/DSig/SHA1_1_0.html`.
31. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. (2002)
32. Artigas, M.S., López, P.G., Ahullo, J.P., Skarmeta, A.F.G.: Cyclone: A novel design schema for hierarchical dhts. Peer-to-Peer Computing, IEEE International Conference on **0** (2005) 49–56
33. Ganesan, P., Gummadi, K., Garcia-Molina, H.: Canon in g major: designing dhts with hierarchical structure. In: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on. (2004) 263–272
34. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. SIGCOMM Comput. Commun. Rev. **34**(4) (2004) 353–366
35. Wu, J.: Handbook On Theoretical And Algorithmic Aspects Of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks. Auerbach Publications, Boston, MA, USA (2005)
36. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM (2003) 175–186
37. Yang, B., Garcia-Molina, H.: Efficient search in peer-to-peer networks. In: Proc. 22nd International Conference on Distributed Computing Systems, 2 - 5 July, 2002, Vienna, Austria, IEEE Computer Society (1998)

38. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: ICS '02: Proceedings of the 16th international conference on Supercomputing, New York, NY, USA, ACM (2002) 84–95
39. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on. (2002) 23–32
40. Cai, H., Ge, P., Wang, J.: Applications of bloom filters in peer-to-peer systems: Issues and questions. In: NAS '08: Proceedings of the 2008 International Conference on Networking, Architecture, and Storage, Washington, DC, USA, IEEE Computer Society (2008) 97–103
41. Bawa, M., Manku, G.S., Raghavan, P.: Sets: Search enhanced by topic segmentation. In: In SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM Press (2003) 306–313
42. Cohen, E., Fiat, A., Kaplan, H.: Associative search in peer to peer networks: Harnessing latent semantics (2003)
43. Berry, M.W., Drmac, Z., Jessup, E.R.: Matrices, vector spaces, and information retrieval. SIAM Rev. **41**(2) (1999) 335–362
44. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science **41** (1990) 391–407
45. Wong, B., Slivkins, A., Sirer, E.G.: Approximate matching for peer-to-peer overlays with cubit (Dec. 2008)
46. Tsoumakos, D., Roussopoulos, N.: Adaptive probabilistic search for peer-to-peer networks. In: Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference on. (Sept. 2003) 102–109
47. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator `http://peersim.sf.net`.