Universidad Rey Juan Carlos
Departamento de Ingeniería Telemática y Tecnología Electrónica

Doctoral Thesis

Self-Adaptation Mechanisms for Efficient Resource Location in Peer-to-Peer Systems

Luis Rodero Merino
lrodero@gsyc.es

Supervisors:

Antonio Fernández Anta
anto@gsyc.es
Luis López Fernández
llopez@gsyc.es

Móstoles, July 2007

I authorize the defense of the Doctoral Thesis *Self-Adaptation Mechanisms for Efficient Resource Location in Peer-to-Peer Systems* written by Luis Rodero Merino

Thesis Supervisor

Antonio Fernández Anta
Madrid, May , 2007

I authorize the defense of the Doctoral Thesis *Self-Adaptation Mechanisms for Efficient Resource Location in Peer-to-Peer Systems* written by Luis Rodero Merino

Thesis Supervisor

Luis López Fernández
Madrid, May , 2007

DOCTORAL THESIS:	*Self-Adaptation Mechanisms for Efficient Resource Location in Peer-to-Peer Systems*
AUTHOR:	Luis Rodero Merino
SUPERVISORS:	Antonio Fernández Anta
	Luis López Fernández

The committe named to evaluate the Thesis above indicated, made up of the following doctors:

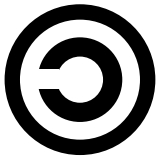
PRESIDENT:	Name Surname
MEMBERS:	Name Surname
	Name Surname
	Name Surname
SECRETARY:	Name Surname

has decided to grant the qualification of:

Móstoles (Madrid, Spain) , 2007

The secretary of the committee.

Luis Rodero Merino
Departamento de Ingeniería Telemática y Tecnología Electrónica
Universidad Rey Juan Carlos,
Móstoles 28933 Madrid



Dedicado a mis padres y a mi hermana, por su paciencia, cariño, y apoyo constantes.

Cumpliendo con mi oficio
piedra con piedra, pluma a pluma,
pasa el invierno y deja
sitios abandonados
habitaciones muertas:
yo trabajo y trabajo,
debo substituir
tantos olvidos,
llenar de pan las tinieblas,
fundar otra vez la esperanza.

Pablo Neruda
(Encontrado en *Pedrá*, de Extremoduro)

Acknowledgments

I would like to thank to my supervisors, Antonio Fernández and Luis López, for their guide, help and support during these years. Also, for they trust and faith in my work (sometimes, greater than my own).

I wish also to thank all the members of the GSyC group for being excellent colleagues, always ready to give a hand at need. I have learnt many things working with you, not only about computing but also about honesty and the search for academic excellence. You have my admiration and respect, apart from my friendship.

To my family and friends... I don't have words to express my gratitude. You have been there from the beginning of this work, with patience and words of encouragement. Probably I wouldn't have been able to finish this thesis without you. My parents and the rest of my family have been a fantastic support (specially my sister, able even to forgive me for becoming a teacher and thus joining 'the dark side' ;-)). Moreover, I am lucky enough to have friends that not only behave as such, but are also an inspiration because of the passion and decision they devote to their own professions and activities.

This work carries a bit, a lot, of each one of you, and I'm greatly proud of that.

Abstract

This doctoral thesis introduces the research work of the author in the field of adaptative topologies applied to the problem of resource location in *Peer-to-Peer* (P2P) networks.

One of the main open issues in P2P systems research is how to efficiently locate resources. Recent works conclude that searches by *random walks* can outperform, in terms of search duration and/or success rate, other mechanisms like flooding, also demanding less resources (bandwidth and processing power). At the same time, they do not have to face the drawbacks of typical structured systems. It has been also observed that the efficiency of random walks is strongly related with the topology of the *overlay network*.

The main contributions of this thesis are:

- An analytical model able to predict the performance (e.g., how long will it take to locate resources) of a P2P system that uses random walks. This model only requires knowledge of the distribution of the nodes degrees and their capacities. With it, we will able to reason which topologies offer a better behavior depending on the load on the system.

- Introduction of *DANTE*, a proposal of a P2P system able to adapt its topology to the load conditions. This self-adaptation behavior emerges from the individual work of nodes, that run a *reconnection mechanism* that does not require neither the intervention of some central coordinator nor global information (as they are not available in pure P2P systems). The adaptation process aims to form topologies as centralized as possible, avoiding at the same time overloading well-connected nodes and preventing sharp changes on the topology. By simulations we will study the behavior of DANTE under different circumstances. Additionally, we will compare it with *GIA*, another solution (well known by the research community) that also implements a topology adaptation mechanism to improve searches efficiency.

Resumen[1]

Esta tesis doctoral presenta el trabajo de investigación del autor en el campo de las topologías adaptativas aplicadas al problema de las búsquedas de recursos en redes *Peer-to-Peer* (P2P) no estructuradas.

Uno de los problemas principales en la investigación sobre redes P2P es la localización eficiente de recursos. Investigaciones recientes indican que las búsquedas por *caminos aleatorios* pueden proporcionar muy buenos resultados (en términos de tiempos de búsqueda o tasa de éxitos) consumiendo menos recursos que otras soluciones como inundación, y sin tener que asumir la rigidez propia de los llamados sistemas estructurados. También se ha determinado que la eficiencia de este tipo de búsquedas depende fuertemente de la topología de la *red superpuesta*.

Las aportaciones básicas de esta tesis son:

- Desarrollo de un modelo capaz de predecir el rendimiento de una red (e.g. cuánto tardarán en procesarse las búsquedas) que use caminos aleatorios para localizar recursos, a partir de la distribución del grado de los nodos y de la capacidad de proceso de los mismos. Esto nos permitirá razonar qué topologías ofrecen un rendimiento más eficiente según las condiciones de carga en la red.

- Presentación de *DANTE*, un sistema P2P capaz de adaptar su topología a las condiciones de carga. Esta auto-adaptación se produce como resultado de la actividad individual de los nodos, que ejecutan un *mecanismo de reconexión*. El proceso de adaptación intenta formar topologías tan centralizadas como sea posible, evitando al mismo tiempo sobrecargar nodos bien conectados o cambios bruscos en la topología. Usaremos simulaciones para estudiar el rendimiento de DANTE y compararlo con *GIA*, otra solución (bien conocida por la comunidad académica) que también usa adaptación para mejorar la eficiencia de las búsquedas.

[1]El apéndice A contiene un resumen extendido en castellano.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this introductory chapter we describe the context of our research: resource location in unstructured *Peer-to-Peer* (P2P) networks, along with the contributions of this thesis to the field.

1.1 Scenario

Peer-to-peer systems have obtained great popularity during the latest years, and they can possibly be considered as one of the most important revolutions happening in the Internet today [68]. P2P is a new communication paradigm where resources such as media files, services, etc., can be both provided and consumed by all participants (also called *peers* or *nodes*). This contrasts with the traditional client-server approach, where the role of each participant is restricted and well defined. In P2P systems, however, each peer is at the same time a server, because it offers resources, and a client, because it requests them. P2P systems are used or proposed for tasks such as ([8]):

- *Collaboration systems.* These systems are aimed to ease the communication among peers, like instant messaging applications such as Jabber [2, 1].
- *Systems for Internet services support.* Systems based on an underlay P2P architecture that provide a certain Internet service. Examples are systems for multicast communications [62] or security applications [44].

- *Distributed Computation.* The goal is to split a functionality that demands high processing capacity into smaller items that are sent to the system peers to run them (*e.g.*, the Seti project [3]). A central entity that coordinates this process, sending the tasks to the peers and gathering the results, is required (Grid computing aims to provide an infrastructure such as that, and so there are authors that see a convergence between P2P and grid systems [30, 38]).

- *Content Distribution.* Systems that allow to publish and share content (*e.g.* media files). In some of them, peers only exchange files directly (Gnutella [77], Kazaa [78]). Other systems create a distributed storage medium to add, replace and retrieve data (OceanStore [49], FreeHaven [24], Mnemosyne [36]).

Compared with classical systems, P2P networks have some advantages. They are more flexible and fault-tolerant, as they are not dependant on a single server node (that would introduce a single point of failure). Another advantage is their scalability, more nodes in the system increase the demand for resources, but also bring a greater offer of them. These properties are mainly due to the lack of a central entity that coordinates or controls the peers. Nonetheless, this same lack of a central coordinator has brought new technical challenges to be solved.

In particular, the problem of *efficient resource location* has driven many efforts from the research community. To allow for resource location, nodes in a P2P system are connected by links, that form an *overlay topology.* When some peer requires a resource, it builds a search message that traverses the overlay network, looking for peers where this resource is offered. The *search mechanism* implemented by the P2P system dictates how search messages are routed through the network.

Roughly speaking, search mechanisms can be classified [8] as either *structured* or *unstructured.* Unstructured networks are less efficient, but on the other

hand they have a greater flexibility that can make them more suitable for systems with high churn of peers (peers enter and leave the system at a high rate), which is a typical scenario of real-world networks [69] (*e.g.*, P2P systems for sharing of media content).

Because of the good properties of unstructured networks, researchers have considered worth to devote important efforts to improve their performance [54]. Search mechanisms based on *random walks* have gained growing attention [31]. The idea of random walks is very simple. Each time a node routes a search message, it chooses the new peer to send the message to uniformly at random among its neighbors in the overlay topology.

Random walks are a solution that consumes few resources compared to flooding, and leverages the dependency on special peers (denoted *superpeers*, see section 2.2.3) that is found in networks such as eDonkey [80]. Yet, there is a possible concern that can appear when using random walks: apparently, there are little guarantees of finding the resource looked for when searching in big networks. As a side effect, false negatives can appear with a certain frequency.

To solve this, recent work has suggested to combine random walks with *dynamic topologies* [17]. This proposal is based on the fact that the performance of random walks is strongly dependant on the overlay topology and the congestion on the system [34]. Assuming that each node knows its neighbors resources, then having a subset of well connected nodes makes random walks able to succeed in few hops. On the other hand, the system must avoid to overload those well connected nodes so they do not become bottlenecks of the network. Our goal is to provide P2P systems with the capacity to self-adapt their overlay topology so nodes with high degree appear, making sure at the same time that none of them gets overloaded.

This thesis presents the research work of the author on the field of dynamic topologies in P2P systems.

1.2 Contributions

This thesis tries to contribute to the research on dynamic topologies in two ways. First, it introduces **DANTE**[1], a proposal of an unstructured P2P system with topology self-adaptation capabilities.

DANTE is based on previous research and work. Initially, we implemented a proposal from Cholvi *et. al.* [17], based on Guimerà studies [34], of a topology adaptation mechanism to study its behavior and performance. Our simulation results confirmed that the implementation worked as expected, and that the assumptions about topologies performance were correct (centralized topologies are optimal for low loads, random ones are optimal for high loads). Yet, we detected some limitations in the adaptation mechanism that could affect its performance on P2P systems.

Our goal has been to evolve that proposal to achieve a system more suited to real P2P world systems. DANTE is the result of these efforts. In DANTE the adaptation process is run by a reconnection mechanism that meets the requirements of a P2P network:

- All nodes implement the same functionality.
- No global knowledge is required.
- No central entity or coordinator is needed.

That mechanisms chooses, for each node, to which other peers in the system that node must connect to. Potential new neighbors are chosen using an attachment kernel that computes how attractive each candidate peer is. That attractiveness is computed by means of a metric that takes into account the candidate's degree, capacity and load. Mainly, a greater degree means that the candidate is more attractive (it knows more about the resources in the network), but that is balanced with the fact that no node in the network shall be over-

[1] From Dynamic self-Adapting Network TopologiEs

loaded. The efficiency of DANTE, as well as its behavior under unfavorable conditions (like attacks and high churn of peers), is tested by simulations.

In this thesis we also present a model that tries to improve the understanding of the relationship between the performance of searches run by random walks and the form of the overlay topology.

This model has two different parts. The first part consists of a set of expressions that measures how the *knowledge* about the network evolves as the random walk traverses the network. This knowledge represents the *state* of the random walk, and allows to estimate the success probability of a search at each hop. From that probability we obtain the expected average search length or number of hops needed, in average, for a random walk to find a random peer starting from any other random node. A peer is found when the random walk visits either it or one of its neighbors. This is due to a characteristic of our model: each node can reply on behalf of its networks (this is a typical assumption in works about dynamic topologies and random walks [54, 14, 21]).

The second part of our model is oriented to compute the efficiency of a network, in terms of the average time needed to solve searches, given the degrees and processing capacities of the peers in the system. These expressions need the expected average length of searches to estimate the overall load on the system. That parameter is computed using the equations of the first part of the model.

The goodness and accuracy of both parts of the model were tested by simulations on different kinds of networks.

1.3 Thesis Structure

This thesis is structured as follows.

Chapters 2, 3 and 4 describe the state of the art of the field of resource location in P2P networks. Chapter 2 summarizes the main types of resource location solutions; chapter 3 focuses on random walks related research, including the description of some systems that propose and apply the use of dynamic topologies; finally chapter 4 introduces the work that was the basis of this re-

search: a study about optimal search topologies, and a first proposal to bring that work's conclusions to a P2P system.

Chapter 5 explains the model developed during this research. As explained above, this model tries to predict the performance of random walks on a P2P network, given the degrees and capacities of the nodes of the system.

Chapter 6 contains a description of a real implementation we have developed of a previous proposal of a self-adapting P2P system, that was the inspiration of DANTE. The experiments run with that implementation served to confirm the validity of that proposal, and also to detect its limitations, that were addressed in DANTE's design.

In chapter 7 DANTE fundamentals are presented. In particular, the reconnection mechanism implemented by DANTE's peers, that makes the network to adapt to the load conditions as intended, is explained. Also, the results obtained through simulations are shown and discussed.

Finally, chapter 8 gives the conclusions and possible future lines of work related with our research.

Chapter 2

Resource Location Solutions in P2P Systems

The Peer-to-Peer paradigm provides users with new communication opportunities. P2P systems present advantages like flexibility, scalability and fault tolerance, thanks to the lack of central coordinators or controllers. But this same lack of central entities brings new technical challenges.

One of the key issues to be solved is *how to locate resources efficiently*. Several solutions have been proposed, each with its advantages and drawbacks. Traditionally, the computer science literature has identified the following families of P2P networks:

- **Structured Networks**. These systems use specialized placement algorithms to assign the *responsibility* for each resource to a specific peer, as well as "directed" search mechanisms to efficiently locate these resources.
- **Unstructured Networks**. These systems do not have precise control over the resources placement. Depending on the degree of centralization, we have:
  - **Centralized Systems**. They are based on using a single central server that stores the index of all the resources offered by the peers in the network.

– **Decentralized Systems**. The task of locating resources is performed and shared by the peers of the network. Three resource location mechanisms are traditionally used with these systems: *flooding*, *superpeers* or *random walks*.

In this chapter we will further describe the fundamentals of these solutions, their advantages and drawbacks, along with some proposed, well-known systems.

2.1 Structured Peer-to-Peer Systems

As said before, structured P2P networks assign the responsibility of each resource to a specific peer, meaning that that peer is responsible of 'knowing' where the resource is held. Generally, it keeps a reference that points to the node of the network that really holds the resource. So a key issue in structured networks is how the responsibility is assigned.

Typically, this is done using a space of identifiers. Each node and resource are assigned an ID, usually computed using a hash function (like SHA-1 [83]). For example, to obtain some node ID, the hash function could be applied to its IP address and port. To compute the ID of a resource we could use its name, or its content if the resource is a file.

Besides, each node is responsible of a partition of the space, so it must know the location of all resources whose keys are inside that partition. The system is in charge of mapping keys to nodes, so whenever some user asks for a certain key (referring some resource) the system must return the node that is responsible for that key, and hence knows where the resource is located. Due to this functionality, these systems are also called *Distributed Hash Tables* (DHT).

What differs between distinct structured systems is how the keyspace partitioning is performed, how the keyspace is reassigned when peers enter and leave the network, and how searches are routed through the network. In general, this kind of networks are considered to have some desirable properties:

- They require less communication steps than unstructured networks to find

some resource. Resources are located in $O(\log(n))$ hops, where n is the number of peers. Hence, little traffic due to search messages must be handled by the network.

- Resources are *always* found if they are in the network and the system is in a stable state. That is, there are not *false negatives*: if one resource is not located by a search message, it is because no peer holds it. The only situation where this can fail is when there is a high churn of peers.

Some of the best known proposals from academia are Chord [73], Pastry [67], Tapestry [89], CAN [61], Kademlia [56] and Viceroy [55]. On the rest of this section, we will explain the basis of Chord, CAN and Kademlia, and finally we will describe some of the drawbacks of structured systems, that can make unstructured ones a more suitable choice under certain circumstances.

2.1.1 Chord

Chord [73] (as Pastry and Tapestry) uses a technique strongly inspired on the distributed data location protocol developed by Plaxton *et* al. [60], although, as their creators claim, it is substantially less complicated and handles concurrent nodes joins and failures well. It uses *consistent hashing* [43] to compute nodes and keys identifiers. By using consistent hashing, Chord ensures that keys are fairly distributed among nodes (all receive roughly the same number of keys).

In Chord, identifiers are ordered in an *identifier circle* modulo 2^m, where m is the identifier length in bits. Key k is assigned to the first node whose id is equal to or follows k. That node is called the *successor node* of k, $successor(k)$. If identifiers are represented in a circle, then the successor of k is the first node found in the circle in clockwise direction.

See for example Figure 2.1, where the circle of identifiers for $m = 3$ is drawn. For simplicity, we denote the key with id i as *key i*, and the node with id j as *node j*. In Figure 2.1 there are three nodes and five keys. *Key 2* is assigned to *node 2*. *Key 3* on the other hand, as there is no node with id 3, is assigned

to the next node that is found clockwise, in this case *node 5*. Finally, *key 6* and *key 7* are assigned to *node 0*.

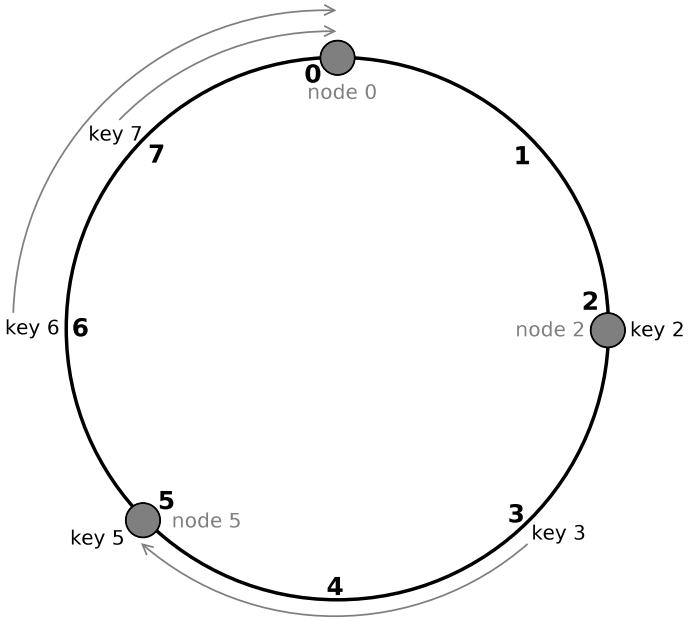


Figure 2.1: Chord ID circle for m = 3 bits.

At the same time, Chord nodes store information to route searches. Routing information is stored in the *finger table*, a table that has, at most, m entries (where m is again the identifiers length in bits). Those entries, denoted fingers, are filled as follows. The i^{th} entry of node n's finger table contains the node, s, whose id is at least at distance 2^{i-1} following the circle of identifiers. That is, $s = successor(n_{id} + 2^{i-1})$, where n_{id} is the id of node n. This can be also explained in terms of id ranges. Each entry in the finger table *represents* a range of the id space. The i^{th} entry in node n's table contains the first node whose id is found in the id circle inside the range of ids that are at a distance between 2^{i-1} (included) and 2^i (excluded) from n_{id} in the id circle, that is, the range $[(n_{id} + 2^{i-1}) \mod 2^m, (n_{id} + 2^i) \mod 2^m)$.

An example can be seen in Figure 2.2. See finger table of *node 5*. Ranges for that node are $[5+2^0 \mod 2^3, 5+2^1 \mod 2^3)$, $[5+2^1 \mod 2^3, 5+2^2 \mod 2^3)$

and $[5 + 2^2 \mod 2^3,\ 5 + 2^3 \mod 2^3)$, that is, $[6, 7)$, $[7, 1)$ and $[1, 5)$. For the first two ranges, the table points to *node 0*, as it is the first node in the id circle after 6 and 7. For the third range, however, the table points to *node 2*, as it is the first node after id 1.

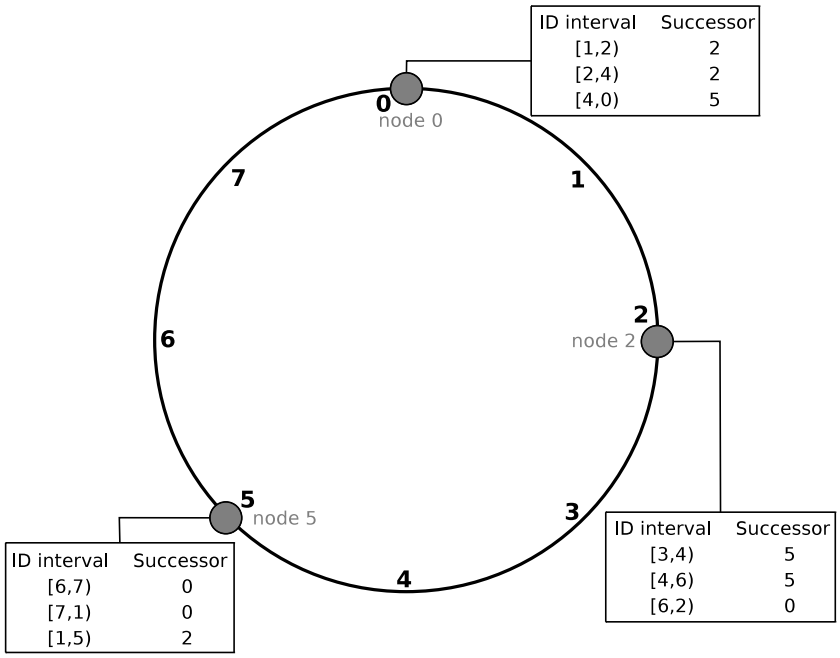


Figure 2.2: Finger tables in a Chord network for m = 3 bits.

To route a search for key k, if the node does not know $successor(k)$, it looks in its finger table for the node j whose id immediately precedes k. Then, it asks j for the node it knows that is closest to k. Repeating this process iteratively, n gets information about nodes closer to k at each step. In [73] the authors show analytically that the average path length is $O(\log N)$, where N is the network size (the value obtained experimentally is roughly $\frac{1}{2}\log_2 N$).

Of course, when nodes enter and leave the network the finger tables must be updated, and keys must be reassigned. The way this process is performed is described in [73], where the authors also use simulations to study Chord's behavior in the face of nodes churn.

2.1.2 CAN: A Content-Addressable Network

CAN[1] [61] is a DHT that models the ID space as a cartesian space of d dimensions. The space is divided into zones (rectangular if $d = 2$, cubic if $d = 3$, etc). Each zone is assigned to one peer. There is not any intersection between zones, and the zones cover all the space (*i.e.* the zones form a partition of the ID space). In Figure 2.3 we can see an example of a simple ID space of 2 dimensions split into 3 zones.

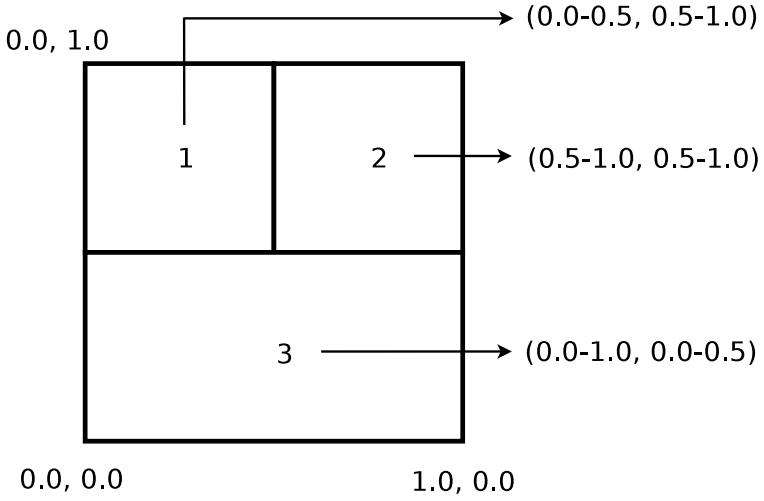


Figure 2.3: Example of a rectangular CAN ID space.

The coordinate space is used to store (key, value) pairs. A (K, V) pair would be stored as follows. The key K is mapped to a point P in the coordinate space using an uniform hash function. The corresponding key-value (K, V) is then stored at the node that owns the zone within which the point P lies. For example, if the K key is mapped to the point $(0.1, 0.8)$, then the peer owner of the zone labeled 1 is the one to hold the pair (K, V). Later, to find the value related with K, the hash function would be applied again to obtain the point P, and then a search message would be routed to the node that owns the

[1]From Content Addressable Network.

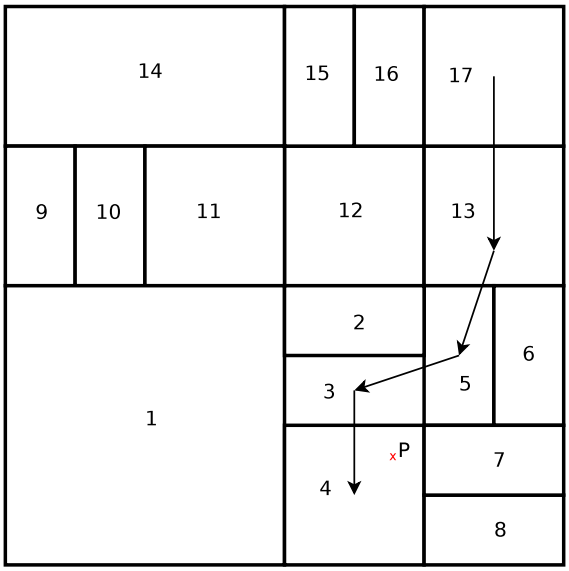


Figure 2.4: Example of search through a CAN ID space.

corresponding zone.

To allow routing, each node maintains a list of peers (IP address and port) whose zones are adjacent to it, its neighbors (in a d-dimensional coordinate space, two zones are adjacent if they overlap in $d-1$ dimensions, and are contiguous in one dimension). Besides, all messages contain a destination point P_d inside the coordinate space. The routing mechanism is simple: each node always routes messages to the neighbor whose zone is closer to the destination point P_d. Thanks to this routing mechanism, the average path length is $O(n^{1/d})$. Figure 2.4 shows an example of how a search message is routed through the coordinate space, until it reaches the zone where the destination P lays.

Further information about how CAN networks are built, and how they handle nodes departures and recoveries (which involve zones partitioning and merging) can be found in [61].

2.1.3 Kademlia

Kademlia [56] is one of the latest structured Peer-to-Peer networks to be proposed, but has already gained a lot of attention. In fact, the latest versions of clients of the eDonkey [80] network (such as eMule) also work as clients of Kademlia-based networks, (two in particular: Overnet [81] and Kad). It is the first structured system to have reached such acceptance in a worldwide file-sharing community. Nonetheless, it has not been studied yet how many searches are indeed solved by the Kademlia-based networks instead of the eDonkey network, and how the efficiency of searches has improved since the incorporation of Kademlia protocols. Despite of this, Kademlia seems a very promising solution.

Kademlia is a DHT based on an innovative idea: the use of the *XOR* metric to compute the distance between two identifiers ($d(x, y) = x \oplus y$).

In Kademlia, nodes are organized as the leafs of a binary tree. Each node's position is determined by the shortest unique prefix of its ID. See an example of a binary tree in Figure 2.5, where nodes ID's are four bits long. Then, *each node divides the tree into subtrees, starting from the root and choosing at each step the subtree that does not contain the node*, following the path that leads to the node's ID. See an example in Figures 2.6, 2.7, and 2.8 of how to build the subtrees corresponding to node 1100.

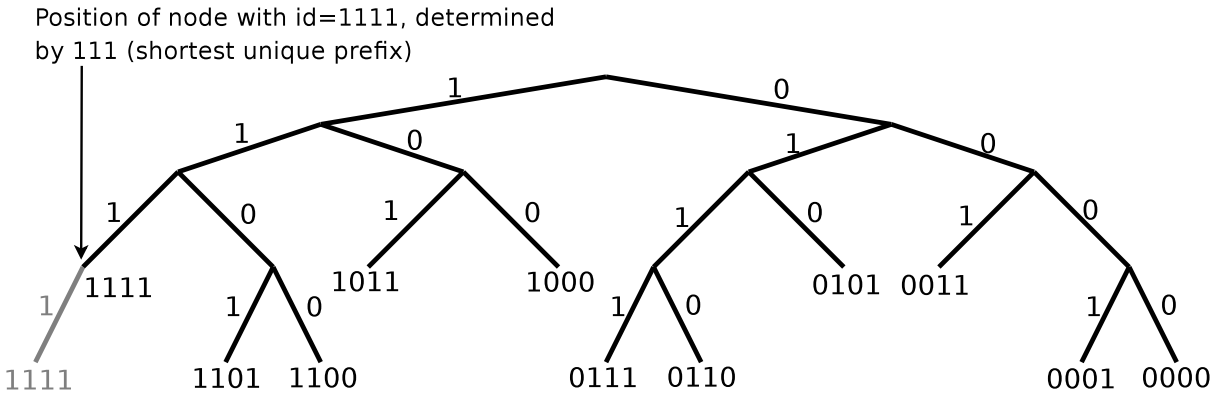


Figure 2.5: Nodes of a Kademlia network in a binary tree.

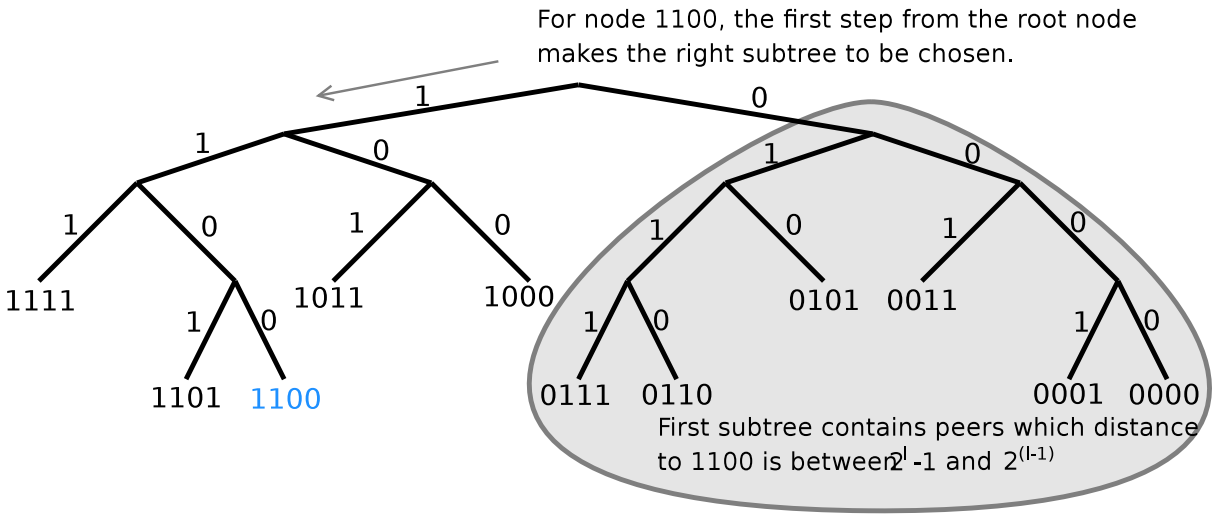


Figure 2.6: First Kademlia subtree for node 1100.

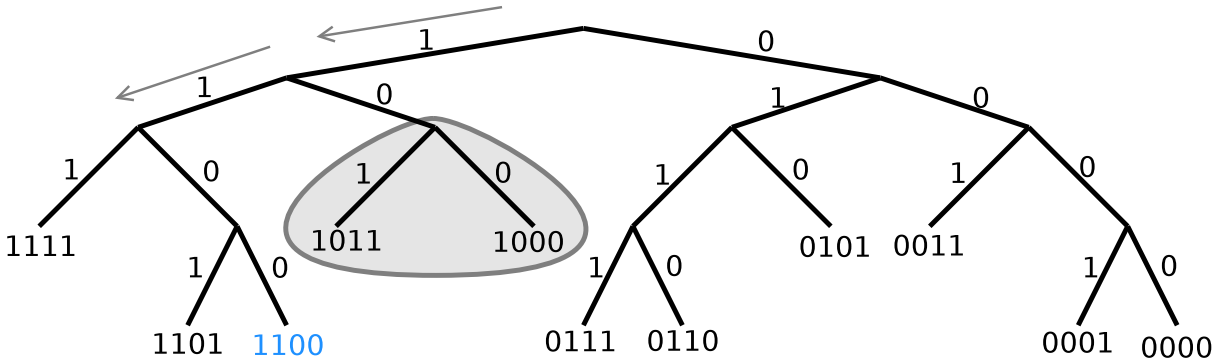


Figure 2.7: Second Kademlia subtree for node 1100.

Let l be the length in bits of the ID's. Then note that each one of the trees built by this process contains the nodes whose mutual distance is between 2^i and $2^{i+1} - 1$, with $0 \leq i < l$. Also realize that, for any given subtree and node x of that subtree, the distance between x and y, where y is also contained in that subtree, is always less than the distance between x and z where z is a third peer from outside the subtree.

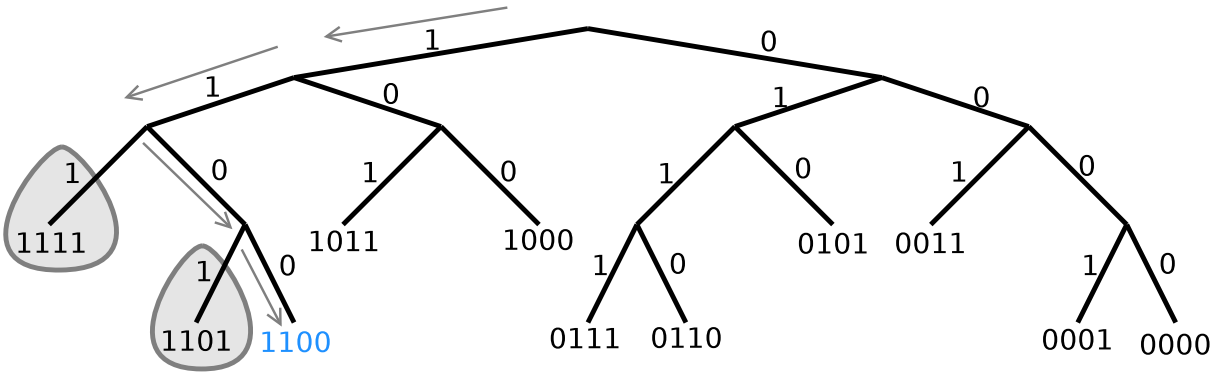


Figure 2.8: Third and fourth Kademlia subtrees for node 1100.

Kademlia ensures that each node knows at least one node in each of its subtrees. For example, node 1100 could know nodes 1111, 1101, 1000 and 0001 in each of its subtrees, as is shown in Figure 2.9.

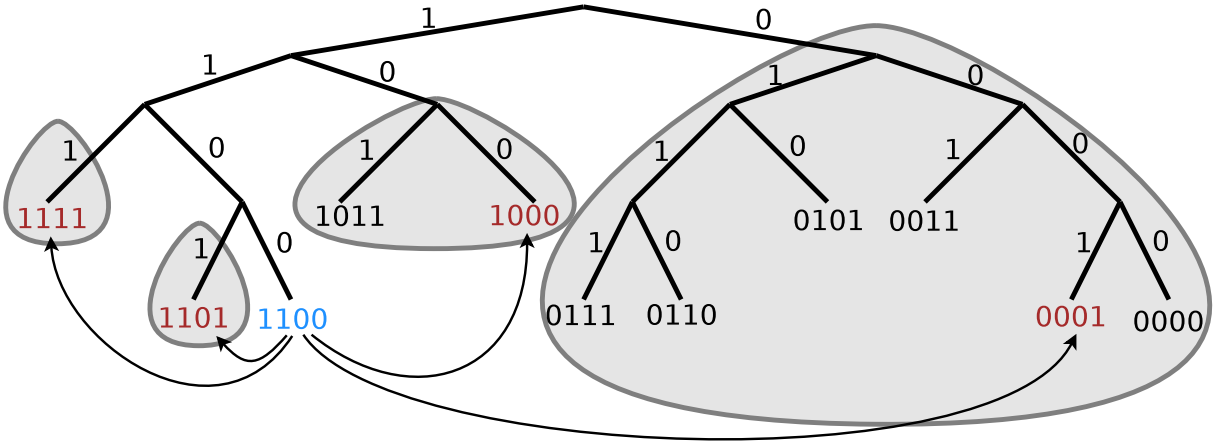


Figure 2.9: Known nodes in each subtree of node 1100.

In Kademlia, the responsibility for each resource is assigned to the node whose ID is closer to the ID of the resource. Then, when one peer wants to locate some resource, it first computes its ID (using SHA-1). Then, it uses the *XOR* metric to compute which of its known nodes (those representing its corresponding subtrees) is closer to the resource, and sends a query to that node.

When receiving the query, the peer (if it does not know the resource itself) will perform the same *XOR* operation over the resource ID and its known peers (which correspond to its own subtrees), and will reply with the peer closest to the resource it knows about. Once this reply is received, the querying node sends the query to that latter node. This process is run iteratively until the node that knows the resource is found. Realize that, at each step, we are reducing the distance to the target by at least 1/2.

For example, node 1100 wants to locate a resource with ID 1010. Applying the *XOR* metric with all its known nodes it gets:

$1010 \oplus 1111 = 0101 = 5$
$1010 \oplus 1101 = 0111 = 7$
$1010 \oplus 1000 = \mathbf{0010} = \mathbf{2} \Leftarrow$ Node 1000 is the closest one.
$1010 \oplus 0001 = 1011 = 11$

Then, it will send a query to node 1000. Node 1000 will reply with the ID of the peer it knows closer to the resource ID: 1011. Finally, the node 1100 sends its query to node 1011, that knows the resource (as is the one closer to it in the whole system) and replies back with the resource location. The process in depicted in Figure 2.10.

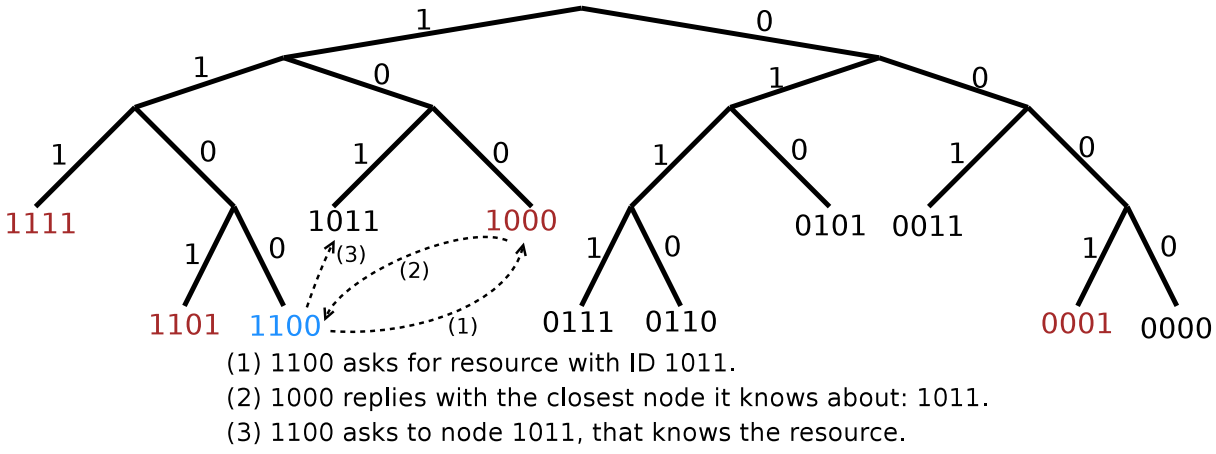


Figure 2.10: Search of resource with ID 1010 from node 1100.

In fact, this is a simplification of how Kademlia works. Kademlia peers do

not know only one node of each subtree but a list of them. These lists are called k-buckets, and are sorted by the last time each node was 'seen' (a message was received from it), with the most recently seen node at the tail. To keep their k-buckets updated, nodes use the IDs of the peers from which they receive any message (request or reply). When some message is received, if the corresponding k-bucket is not full, the ID of the sender node is inserted at the tail. But if the list is full, a ping message is sent to the least recently seen peer (the oldest one, that is at the head of the list). If that peer fails to reply then it is removed from the k-bucket and the request sender node is inserted at the tail of the list. If, however, it replies back, then that node is moved to the tail and the request sender peer is just not added to the list. As we see, Kademlia never removes from the lists those nodes that are alive. The goal is to keep track of old nodes, as the longer one node has been alive, the more likely it will remain alive in the future [68].

Information of how the departures and entries of peers is handled in Kademlia can be found in [56].

2.1.4 Loosely Structured Systems, FreeNet

Loosely structured systems are a special type of structured systems where the peers estimate where is more likely that the resource will be found to route searches. The routing algorithm uses a heuristic, based on local information, and does not guarantee that the resource will be located. A well-known loosely structured network is FreeNet [18, 19].

FreeNet is a P2P system that focuses on providing privacy and anonymity to providers and consumers of information. The goal is to make the network resistant to censorship, while keeping a good performance in terms of availability, efficiency, etc.

To allow for more efficient searches, FreeNet uses a routing algorithm based on IDs. Each node has an unique ID, (also called keys) and the same applies to resources in the network. Additionally, peers keep a table of their neighbors and

their IDs. When a search message arrives to the node, it compares the resource ID with those of its neighbors, and forwards the query to the one whose ID is closest to the resource ID.

This routing mechanism reminds those used by structured P2P systems. Yet, there is no tight control in FreeNet on the neighbors each node must be connected to. In structured systems, there are mechanisms that ensure certain properties about each node's neighbors (for example, in Kademlia is ensured that each node knows of at least one peer in each of its subtrees.). Those properties are necessary for the routing functionality to always work. In FreeNet there are not such mechanisms.

2.1.5 Drawbacks of Structured Peer-to-Peer Systems

Thanks to their efficient search processes, structured networks have been used in certain P2P systems, for example some file storage solutions like OceanStore [49], Scan [15] (based on Tapestry) or Past [25] (based on Pastry). Nonetheless, they have failed so far to gain wide acceptance in file sharing networks, with the possible exception of Kademlia and its recent incorporation to eDonkey clients.

The reasons that explain this lack of acceptance are the drawbacks that structured systems present. These drawbacks are enumerated (more or less exhaustively) in [8], [14] and [37]. Here we list the most important ones:

- **DHT tables are expensive to maintain**. When peers enter or leave the network, DHT tables must be rebuilt, and this implies some communication costs. If nodes move often from-into the system, the communication overhead becomes relevant. Because of this, structured systems are considered to perform poorly in scenarios where there is a high churn of peers (like for example in file-sharing networks).

- **Resource popularity is unequally distributed**. It has been observed that content popularity can vary strongly in content-sharing P2P networks [35]. The more popular some resource is, the more searches are

done for that particular resource. Hence, nodes that are assigned popular resources will support much more load than the rest of peers. Besides, DHT tables do not take into account node capacities when assigning resources, so it is easy that low capacity nodes have to deal with high load: there is a lack of 'fairness' on the system. Load balancing mechanisms have been developed to deal with this problem [32].

- **No advantage is taken from natural replication**. It can be expected that, the more popular is some resource, the more peers will download and cache it. That is, resource replication is increased by resource popularity in a natural manner. Then, it should be easier to find popular contents as more nodes keep them, and load should be more fairly balanced. But DHT fails in taking advantage of this, as queries are always addressed to the particular node to which the DHT table assigns the resource. To solve this, some authors propose using a cache that stores results of queries as they traverse the network, so more popular resources will then be cached with high probability. Kademlia (see section 2.1.3), for example, uses a cache mechanism.

- **Typical searches are by keyword, not by exact match**. Hash functions can only be applied on exact resource identifiers, but users often only can search by keyword, as they do not know the exact resource name, and/or wish to know all the resources that contain some keyword in their name or content. It is possible to provide keyword search on structured networks, typically using an inverted index by keyword [63], but maintaining these indexes can be expensive in terms of communication and processing costs, as any term can be a keyword.

- **Locality is destroyed**. Data items locality (data from the same site) can be used to improve searching and browsing efficiency. But in a structured network that locality is lost as resources from the same site are not usually collocated (DHT ignores the resource 'origin' to compute its location).

SkipNet [37] is a proposal of a P2P structured network that preserves locality.

- **Application level information is lost**. Data used by applications is often organized in a hierarchical fashion inside a namespace. That is, the position inside the hierarchy is relevant. But when translating the resource name to a key that information is lost.

2.2 Unstructured Peer-to-Peer Systems

Unstructured P2P networks have no precise control over the resource placement, nor they have a tight control on the way nodes connect each other: potentially, each peer can be connected to any other peer in the network. In general these systems present certain interesting features:

- Have little communications overhead due to the management of the entries and departures of peers. Hence, they can handle situations where the population of peers is highly-transient.

- Take advantage of the spontaneous replication of popular content.

- Allow to perform queries by keywords in a quite simpler way than with directed search protocols.

Because of the properties of unstructured systems they have been considered to be very suitable for mass-market distributed resource sharing.

As it was told at the beginning of this chapter, there are two types of unstructured P2P systems: centralized and decentralized. Decentralized systems use search mechanisms based on *flooding*, *super-peers* and *random walks*. In the next sections the centralized, flooding based and super-peer based systems will be further described. Random walks are explained in chapter 3. A comparison of search methods for unstructured P2P networks can be found in [76].

2.2.1 Centralized Systems

In this kind of systems there is one central server that is in charge of keeping an index of the resources held by peers. At start time, each peer notifies to the central server that it is alive, and sends it the list of resources it offers to the rest of the network. Whenever a peer wants to locate some resource, it just sends the query to the central server, that will reply with the list of nodes that have that resource. As we see, peers have the role of clients of the central server (this can be observed in Figure 2.11).

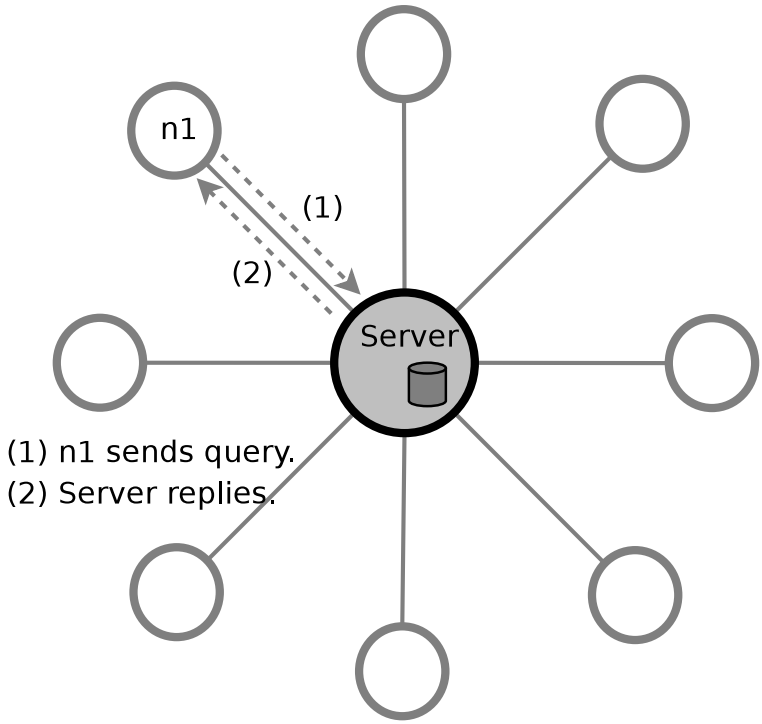


Figure 2.11: Search in a Centralized P2P system.

These systems are also called *hybrid systems* [8] as they combine a centralized solution for resource location with a P2P approach for resource sharing.

The main advantage of centralized systems is that they are simple to implement, and the search mechanism is fast and efficient. Yet, they have the

same disadvantage of any centralized system: *they have a single point of failure*, and so are vulnerable to attacks to the server, censorship, technical failures, etc. Furthermore, these solutions are inherently non-scalable, and limited by the capacity of the central server. A well-known example of a centralized Peer-to-Peer system is Napster [79].

2.2.2 Flooding Search Mechanism

The flooding approach was used by the first content-sharing systems like Gnutella [77]. The idea is simple: each time one peer receives a search message, if checks if it knows the resource looked for. If it is so, a reply message is sent back to the search origin node. If not, it forwards the search message to ALL its neighbors (except the one from which it received the search, see Figure 2.12). This mechanism is also known as *Breadth-First-Search* (BFS). Search scope is limited by a TTL. The TTL is decreased at each hop, and when it reaches 0 the search message is discarded.

It is clear that, the greater the TTL, the higher is the probability of finding the resource as the search message will traverse more nodes. On the other hand, setting a high TTL can lead to the collapse of the network due to the high amount of bandwidth consumed by search messages. In order to decrease the traffic, peers can keep a list of recently forwarded search messages, so if some search reaches them more than once, they can discard it. Yet, this can not alleviate enough the high demand of bandwidth inherent to the use of flooding.

It is widely accepted that flooding based systems can not scale [66], [41]. A brief study should convince us of this. Let t the TTL, $g+1$ the number of neighbors of each node (so at each hop the search is forwarded to g neighbors). Then, an unsuccessful search will cause m messages, where m is computed as:

$$m = (g+1) + (g+1)g + (g+1)g^2 + \ldots + (g+1)g^{(t-1)} \qquad (2.1)$$

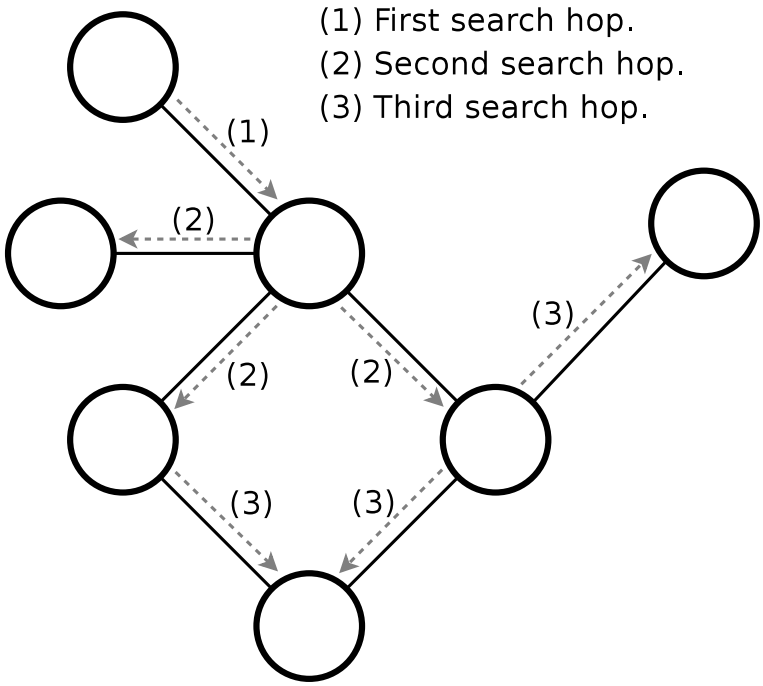


Figure 2.12: Search in a Gnutella-like network.

that can be simplified[2] to:

$$m \;=\; (g+1)\sum_{i=0}^{t-1} g^i \;=\; (g+1)\frac{g^t-1}{g-1} \;=\; \Theta(g^t) \tag{2.2}$$

So, for example, if we set $g = 5$ and $t = 5$, we have that $m \approx 4700$.

Let N be the network size, and λ the number of searches each node starts per unit of time. Finally, let p the average proportion of messages that are not created because the search is successful at some hop, or the message has already traversed that peer and so is not forwarded. Then, the total load in the network M is defined as:

$$M \;=\; N \;\times\; \lambda \;\times\; m(1-p) \tag{2.3}$$

As the network size grows the consumed bandwidth grows linearly with the number of nodes (assuming that g keeps constant). But, what is worse, in order

[2]The sum of a geometric progression is :$\sum_{i=a}^{b-1} c^i \;=\; \frac{c^b-c^a}{c-1}$

to cover a certain proportion of the network, the TTL also needs to be increased and so the number of messages will grow exponentially.

Improved Alternatives to Flooding

There are some works that try to improve the performance of Gnutella, applying certain changes to the pure BFS algorithm.

There is an interesting proposal [86] from Yang and Garcia-Molina that tries to refine the flooding technique by different approaches:

- *Iterative deepening*, where multiple breadth-first searches are initiated sequentially with increasing depth limits until the resource is found or the maximum TTL is reached (this same method is called *expanding ring* in [53]).

- *Directed BFS*, where the source peer sends the query only to those nodes that it deems quality results can be received, neighbors then will forward the networks using typical flooding.

- *Local indices*, where nodes keep an index of all peers within r hops from itself, and searches are processed only at the depths specified by a policy.

In [42] we find another work targeted to improve the results of the flooding technique. They propose two different modifications to BFS:

- Each peer forwards the search message only to a subset of its neighbors. With this simple technique, the network is explored using less messages than the brute force flooding method.

- *Intelligent Search Mechanism.* Peers keep a *profile* of its neighbors, based on the successful queries that were returned by each one of them. These profiles are used to rank the neighbors. When a search is received, the node decides using that ranking which of its neighbor are more likely to return a relevant result to the query.

Finally, the *Distributed Resource Location Protocol* (DRLP) is introduced in [57]. By this routing mechanism, when a search message arrives to some peer a, which does not know the resource, the node will forward the search to each of its neighbors (except the sender) with probability p. This is similar to the first modification proposed by [42], that was described above. Additionally, when a resource is found by a search message, a *resource found* message is sent to the query-originating source, following the same path that the search message traversed. All the intermediate nodes that are crossed by the *resource found* message store in their a cache the location of the resource.

The authors of [57] also make an analysis of the efficiency of DRLP that concludes that by this approach searches finish successfully with a high probability, while the number of messages used is a small fraction of the number of messages that flooding would require. Nonetheless, there are two problems that are left open: how to compute p (as the replication of each resource is mostly unknown), and the limited size of the caches.

In general, simulation results show a better performance of these proposals compared with the basic flooding mechanism: they require less messages, while the size of the profiles is kept low.

These solutions help to reduce the overall resource consumption (bandwidth and processing power) of flooding, however they can not solve the problem of scalability as, at the end, they still use BFS as the basic search technique. Assuming, for example, that nodes forward searches to only a certain average rate r of their neighbors. Then, Equation 2.2 would become:

$$m \;=\; r(g+1)\frac{(rg)^t-1}{rg-1} \;=\; \Theta((rg)^t) \tag{2.4}$$

So, at the end, the grow of traffic keeps being of the same order than before. Because of this lack of scalability, other proposals have appeared like those based on superpeers. Nowadays those networks have replaced Gnutella-like systems in popularity.

2.2.3 Superpeers Search Mechanism

The superpeers based solutions are, in fact, not 'pure' P2P systems. Here, we find at least two different roles: *normal* nodes (also called *ordinary* nodes in the literature), that provide and demand resources, and superpeers, that keep indexes of resources. All normal nodes are connected to at least one superpeer. Each superpeer keeps the index of resources held by all the normal peers connected to it. Besides, superpeers can be connected among them, forming a subnet that is the core of the P2P system. See for example Figure 2.13, where normal nodes (white nodes) are all connected to some superpeer (in grey).

There are two important circumstances that these networks benefit of:

- Discovery time is considerably reduced compared with other implementations (like flooding), while there still is no single point of failure.

- The heterogeneity of nodes is taken into account, so most capable nodes will become superpeers and so they will perform more tasks than normal nodes.

Heterogeneity is, indeed, a factor P2P networks could benefit of. In real P2P systems nodes will have different processing power, bandwidth, storage capacity, etc. But the P2P paradigm initially assumes all members to be 'equal', in the sense that they must share responsibilities (functionality, load, etc) regardless of their capacities. Using superpeers is a first approach to take advantage of the heterogeneity of the peers in the system, so the most capable ones are in charge of most of the work.

But, on the other hand, these systems have certain vulnerabilities. If superpeers fail (they are attacked, closed by censorship, etc), other nodes must take their place so that the P2P system can keep working. Even in the case that those peers are available, it could happen that some of them do not have enough capacity. Those new superpeers would become a bottleneck that could decrease the system performance.

This solution reminds of centralized systems. Yet, there is an important difference: there is more than one superpeer, so in case one fails others can take its place. Besides, any node should be able to play the role of a superpeer if it is demanded so and it has enough capacity and bandwidth (and the owner of the peer is willing to offer it as a superpeer, in some cases P2P software can allow users to disable this feature).

Many different systems can be designed based on this simple idea, differing for example in the way that superpeers interact. In Figure 2.13 we see a system where superpeers forward their queries to other superpeers. Yet, it must not be so in all cases, and in fact real superpeers based systems implement different approaches. For example, superpeers in the eDonkey [80] network do not interact at all.

One of the first academic proposals based on groups of peers around special nodes is *CAP* [48] (Cluster-based Architecture for P2P). In CAP, nodes are grouped in *clusters*, where a cluster is a set of peers that are topologically closed. Each cluster has a *delegate node* that stores the resources held by the members of the cluster. When performing a search, nodes only ask to the delegate of their group. If the delegate does not know the resource, the request is sent to other delegates. It is easy to recognize in CAP the concepts of super-peer networks. Yet, CAP has some important limitations. It depends on an unique *clustering server* that is used to build the groups (when one node joins the network, the server tells it to which group it belongs and the corresponding delegate). Besides, when choosing the delegate, considerations about peers capacities are not taken into account.

In [85][3], Yang and Garcia-Molina perform an analytical study and comparison by simulations of different policies for super-peer networks design and organization. The same authors introduce in [87] an analysis of a particular type of super-peer network where super-peers forward queries to other super-peers by

[3]The title of the paper *Comparing Hybrid Peer-to-Peer Systems*, can be misleading. Hybrid is usually applied to systems with only one server (e.g., Napster), yet the authors study systems with several interconnected servers.

flooding. From that model they infer some *rules of thumb* for networks design. Unfortunately, it is not clear that their conclusions can be easily translated to other super-peer systems with different behavior.

Finally, it is worth to note the high acceptance of super-peers based systems for file-sharing activities. Some real, widely used networks are Kazaa [78] (based on the FastTrack protocol) and eDonkey [80] (based on the protocol of the same name). How peers interact in the Kazaa network is not totally known. The reason is that the company that created Kazaa (Sharman Networks Ltd.) never published the specifications of the system (communications protocol and peers behavior). Besides, the software developed is closed, and most of the communications among peers are encrypted. Yet, there are some efforts that try to improve the understanding of the Kazaa system like [52] and [51]. The eDonkey protocol is, on the other hand, well known and documented [50].

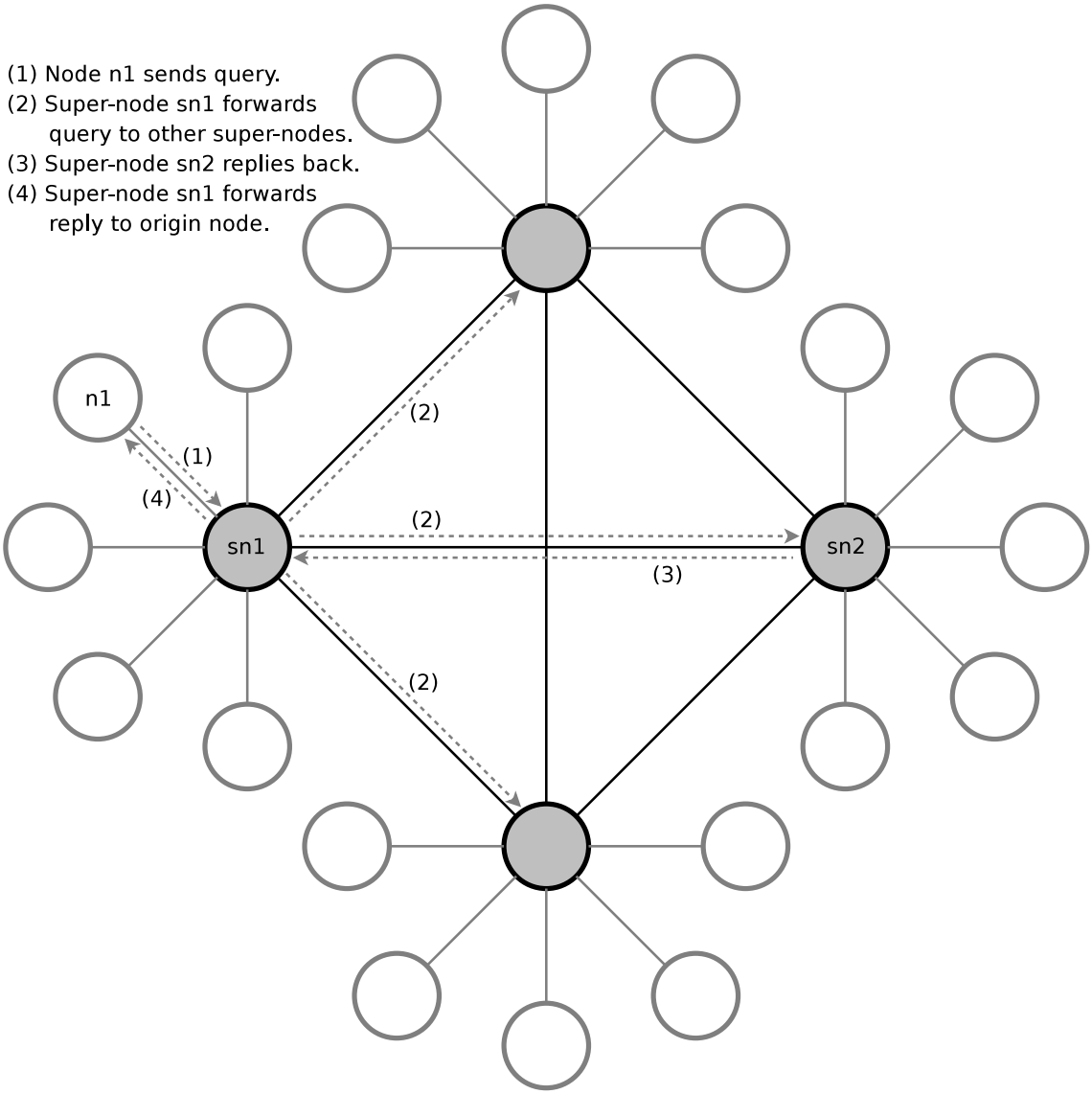


Figure 2.13: Search in a superpeers network.

Chapter 3

Resource Location by Random Walks

Random walks is a technique used for the problem of resource location in P2P systems that has gained a growing attention from the research community. The basic idea is simple, given some node a in a graph, a random step is a move from a to some node b, where b is a uniform and randomly chosen peer from the set of neighbors of a. A random walk is a sequence of these random steps, starting from some initial node (see for example Figure 3.1).

Resource location by random walks just uses search messages that follow a random walk through the network until the resource is found or the message is discarded by some mechanism (like a bounded TTL). Those search messages are also called *walkers*. Some works [29] assume that nodes can send not only a walker when starting a new search, but k (being k a small number). The literature denotes this as *k-random walk*.

The random walk solution can seem somewhat *naive*, and immediately raises questions about its efficiency:

- There is not guarantee that the resource will be found, so *false negatives* can happen. In fact, it seems that in some scenarios resources will not be found with a high probability.

- It can take a long time to find a resource, depending on the size of the

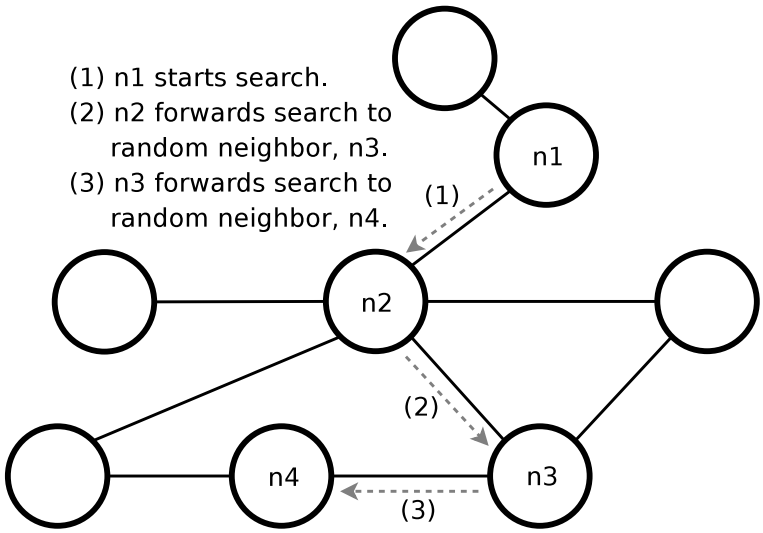


Figure 3.1: Search by random walk.

network, the degree of replication of the resource, the capacity and load on the peers traversed, the network topology, etc.

Yet, there are works that point out that random walks can be more efficient that other solutions like flooding. Specially interesting is the work of Lv *et al* [53], introduced in the next section.

3.1 About the Efficiency of Random Walks

In [53] the authors study and compare different aspects of the behavior of flooding and k-random walkers in different scenarios. These scenarios are set by:

- *The network topology.* Four different topologies are studied: *random graphs*, *power-law graphs*[1], *gnutella graphs*[2], and *two-dimensional grids* of 100×100 nodes.

[1]Topology where the nodes degrees follow a power-law distribution. That is, the probability for any node of having k neighbors is proportional to k^{-i}, where typically $2 \leq i \leq 3$.

[2]As obtained empirically in October 2000. It follows a two segments power-law distribution.

- *Resources popularity.* The *popularity* of a resource is the probability that it will be requested by the users. Formally, the resource i has popularity q_i if the probability of any new search to be for resource i is q_i. Let m be the number of different resources in the system ($i \in \{1, ..., m\}$), then $\sum_{i=1}^{m} q_i = 1$. In particular, in [53] the authors study two different popularity distributions: *Uniform*, where all objects are equally popular, and *Zip-f*, where popularity follows a Zipf law. This distribution is relevant as different studies show that Napster and Gnutella queries follow Zipf-like distributions [71].

- *Resources replication.* The *replication* of a resource refers to the probability that any randomly chosen node knows the resource. The replications distributions used by the authors are *Uniform*, *Proportional to popularity* and *Square-root* (see 3.2.2 for a description of these strategies).

From the analysis of their experiments, Lv *et al.* conclude that flooding incurs in considerable overhead, making it unable to scale. The use of *expanding ring* as the search mechanism (similar to the *iterative deepening* approach explained in section 2.2.2) saves a certain amount of overhead, yet it does not solve the issue of multiple copies of messages that compromises the scalability of flooding-based systems.

But more importantly, the authors show by simulations in the same scenarios described above that *using k-random walks outperforms flooding by up to two orders of magnitude.* In their experiments they use from 16 to 64 walkers, although the best results were obtained using 32. With that configuration, the random walks approach consistently performs better than flooding in all scenarios, needing many less messages (about two orders of magnitude less). It is true that random walks also need more hops to find the resource, but as the load in the nodes is kept low, they guess that in a real scenario that would lead to lesser effective times, as little load would make nodes to handle messages more quickly.

However, the authors conclude also that *random graphs* are better for searching that *power law* and *Gnutella graphs*, as high degree nodes take much more load and introduce more duplication overhead in searches. This reasoning is due to a characteristic of their model: they assume that all nodes have the same capacity, *i.e.* the network is homogeneous. But in heterogeneous networks, this is not necessarily true [16].

In the work from Gkantsidis *et al.* [31], the authors analyze the performance of random walks for searching and constructing P2P networks. In this work the authors state that random walks have two basic analytical properties:

- The random walk can simulate independent uniform sampling in P2P networks with great accuracy, using the final node after a number of hops as a sample. The number of hops depends on the characteristics of the network.

- The number of hops required to simulate uniform sampling can be as low as the number of independent samplings. This is counter-intuitive due to the dependencies between successive steps in a random walk.

This properties make random walks suitable for searching and network construction, as the authors discuss in their work. They also include simulations where search by random walks, uniform sampling, and flooding are compared. The goal is to measure which method obtains a greater number of hits (load is not an issue authors take into account). As expected, flooding is the one with worst results, and sampling obtains the best performance. Random walks stay in the middle, with results close to independent uniform sampling.

3.2 Proposals to Improve the Performance of Random Walks

There are some works in the literature that try to increase the efficiency of random walks by introducing some changes to the basic walking mechanism. The

pure random walk technique is *blind* in the sense that the next peer in the walk is chosen at random, and no neighbor will be chosen with a higher probability than the others. Besides, nodes do not keep information about previous searches. The proposals explained in this section introduce some changes in order to get better results than the pure random walk.

3.2.1 Informed Searches

Some solutions are based on using information about previous searches, so peers can forward walkers to regions of the network where it is more likely that the resource will be found, or estimate the number of walkers they should use to perform the searches. In this section some of these proposals are introduced.

Record of Previous Searches

This is a generic improvement proposed by Lv *et al.* [53]. By this approach, each peer keeps a record of the searches that have traversed the peer previously (each search must have an unique ID), and to which neighbors each one of those searches where forwarded to. When a search message with the same ID arrives again to the node, the search is forwarded to any other neighbor that it has not been forwarded before. By this simple mechanism, walkers can cover a greater amount of nodes in the same number of hops. Depending on the topology, this can reduce the message overhead and number of hops by up to 30%.

Adaptative Probabilistic Search

In [75], Tsoumakos and Roussopoulos propose a variation of the k-random walk method, called *Adaptative Probabilistic Search* (APS) where the forwarding process is *probabilistic* instead of random.

In APS, peers use feedback from previous searches to redirect the walkers. Each node keeps an index that associates neighbors with probabilities. Whenever a search crosses the peer, and if the peer does not know the resource, it chooses the neighbor to forward the walker to using that index. Besides, it up-

dates the index of the chosen neighbor, increasing it in the optimistic approach (the node assumes the search will be successful) or decreasing it in the pessimistic approach (the node assumes the search will fail). When searches finish, either because the resource was found or because the TTL of all the walkers reached 0, the indexes of the nodes traversed by the search are appropriately updated. For example, in the pessimistic approach, if the search was successful, the nodes in the path of the search must update the indexes of the neighbors they chose increasing them, but if the search failed, the indexes must not change.

By simulations, authors show that APS can obtain higher success rates than pure random walks creating as little overhead as them.

Routing Index

An approach quite similar to APS is taken in [22]. Here, nodes keep a *Routing Index* (RI) that is used to decide to which neighbor the walkers should be forwarded. Hence, as in APS, the purpose of these indexes is to point which neighbor is most likely in the direction of the resource looked for. For each neighbor, the RI keeps a estimation of the total number of resources and the number of resources of certain *topics* (assuming that queries can also be related to topics) that can be found through that path. In [22] the authors explain different ways of how RIs can be managed. Also, by simulations, authors show how the RI mechanism improves the efficiency of flooding or pure random walks.

Equation Based Adaptative Search

Another proposal is described in [10]. In this work, the authors first propose a mathematical model for random walks, which characterizes the performance of searches as a function of the number of random walks, their TTL, and the popularity of the resource being looked for. With these parameters, the model estimates the success rate, the overhead in the network (number of packets traversing the network) and the search delay (in time). Based on this model, the authors then introduce a search algorithm denoted *Equation Based Adap-*

tative Search (EBAS). When a new search is started, the source node uses the mathematical model to compute the number of walkers k and their TTL that, given the replication of the resource, guarantee a target performance. The performance is set as bounds on the success rate, delay, and overhead.

The authors present the results of some simulations that show how this adaptative approach performs better than using pure random walks. Yet, this mechanism has an important drawback: peers must keep an estimation of the replication of resources, calculated from feedback obtained in previous searches, using the algorithm proposed in the work. However, in some scenarios, making all the peers to keep such estimation, even for only a proportion of all the resources in the system, can demand too much storage space and processing capacity for certain nodes with low capacity.

Attenuated Bloom Filters

In [64] the authors propose a modified version of the random-walk, where the search message is forwarded using the information about the resources held by nodes at different distances. That information is stored using a modified version of Bloom filters [11, 13], that the authors call *Attenuated Bloom Filters* (ABF). The main goal of ABF is to reduce the latency to find nearby replicas, and the authors conceived it as a complement to deterministic search algorithms (like Tapestry [89]).

An ABF of depth d is an array of d Bloom filters. Each node a has an ABF associated to each of its neighbor's connections. The filter at position $1 \leq i \leq d$ of the array represents the resources that are i hops away through the corresponding connection. For example, the filter at position 1 of the ABF corresponding to the connection pointing to neighbor b, stores the resources that the node b holds. Besides, each Bloom filter i is assigned a *potential value* $1/2^i$, so the filters representing resources at a lesser distance, are more important.

When a search is processed by one peer, a value is computed for each of that peer's ABFs. That value is computed as the sum of all the potential values for

the levels of the filter which contain the resource looked for. So, if the resource matches the filters at position 2 and 3 of some ABF, the value for that ABF would then be $1/4 + 1/8 = 3/8$. That value is called the *potential value* of the resource for the corresponding ABF. The walker will then be forwarded through the connection with the greatest potential value. If there are two matching ABFs with the same potential value, the connection with the lowest latency is chosen.

Bloom filters can produce false positives. This means that although the ABF matched the resource at level d, after d hops the resource is not found as the corresponding node does not hold it. In that case, instead of using backtracking (sending back the search to the previous node in the path, so it forwards it to its next best neighbor), the authors suggest using a deterministic algorithm. The reason is that ABF is designed to find close replicas if they exist, but using backtracking would transform it in to a kind of Depth First Search that would perform worse than the deterministic algorithm in many cases.

3.2.2 Other Proposals

There are other ideas targeted to improve the efficiency of searches without using information from previous searches. They require less resources from nodes as they do not demand them to keep state.

Adaptative Termination

It is proposed by Lv *et al.* in [53], and introduces a way to limit the scope of searches different from the TTL mechanism. Instead, walkers must query every h hops to the source peer if they must continue or not. By simulation, the authors show that $h = 4$ can give good results. This technique can be combined with the other proposals explained in this section.

Forwarding to Neighbors of Highest Degree

Adamic *et al.* [5], using the generating function formalism for graphs introduced by Newman *et al.* [59], study the efficiency of searches using high degree nodes: basically, search messages are forwarded to the neighbor of highest degree. In this work, searches are for particular peers, where each node knows its neighbors, and the neighbors of those nodes (second neighbors). So when the walker reaches any neighbor of the target node's neighbors the search is finished. The conclusions can be easily translated to a scenario where searches are for resources, and each node knows the resources hold by its neighbors.

The study concludes that using high degree nodes performs better that pure random walks. Adamic work is focused only on power-law networks (that are present in many real world systems, like social or biological networks), but some studies ([41], [40]) have shown that the Gnutella network follows that law, so it is assumed that Adamic conclusions can easily be applied to other P2P systems.

It is important to note that when suggesting to forward searches to high degree nodes the authors do not take into account the overload that can be produced in those peers. Because of this, Adamic conclusions seem to disagree with the work of Lv [53], introduced in Sec. 3.1, that states that high degree nodes should be avoided when building the system topology.

Replication Strategies

The term *resource replication* refers to the fact that some resource i is held by several peers. More precisely, the *replication* of a resource i, r_i, is the number of nodes that hold a copy[3] of the resource. $\sum_i^m r_i = R$, where R is the total number of resources stored in the network, and m is the number of different resources.

How resources are replicated through the network has a deep influence on the search performance. The impact of replication on unstructured networks is

[3]A copy can be an identical copy of the original resource, or just a pointer to the place where the resource is located.

studied in [20]. In that work, Cohen *et al.* define replication strategies that, given the popularity distribution, specify the optimal number of copies of each resource that should be hosted in the network. The goal is to minimize the average search length of walkers. Some replication strategies studied are:

- *Uniform*: $r_i = R/m$. By this strategy, all resources are equally replicated regardless of their popularity.

- *Proportional to popularity*: $r_i \propto q_i$. The "Proportional strategy" makes a number of copies of each resource proportional to its popularity. Usually, this replication is achieved *naturally* in unstructured networks. The more peers demand a resource (and then offer a copy of it), the more copies of the resource will be available. This is sometimes called *Natural Replication*.

- *Square-root*: $r_i \propto \sqrt{q_i}$. Proportional to the square root of the popularity.

Initially, the proportional strategy seems more fitted to achieve good results: the more demanded a resource is, the easier should be to find it, so the more copies should be placed in the network. Nonetheless, this approach has another effect: resources with little popularity will be poorly replicated, and so it will take more jumps to find them.

In fact, in [20] the authors arrive to somewhat surprising conclusions:

- Uniform and Proportional have both the same expected search size on successful queries.

- Uniform and Proportional are the extremes of a family of replication strategies. Any of these strategies have a lesser expected search size than Uniform and Proportional.

- The optimal replication strategy is *Square-root* replication.

Finally, the authors also give some simple distributed algorithms to achieve optimal replication on an unstructured network.

In a different work, Ferreira *et al.* [28] propose a simple replication mechanism using random walks, where each resource is replicated in the latest $\sqrt{N}$ nodes crossed by a random walk with TTL $\log N + \sqrt{N}$. Searches are performed by walkers with a TTL of $\gamma\sqrt{N}+\log N$. Analytically, authors show that searches will finish successfully with high probability even for low values of γ. The idea is based on the birthday paradox.

3.3 Dynamic Adaptative Topologies

Dynamic Adaptative Topologies take a different approach than the previous works to the problem of how to improve the efficiency of random walks. Their goal is to find the topology that makes walkers to find the resource with few hops and little time. These systems have in common:

- *They try to take advantage of the heterogeneity of real networks*, where usually peers have very different capacities like bandwidth, processing power, etc. Thus, adaptative P2P networks aim to build topologies where nodes of high capacity have the biggest degrees and so handle a bigger share of the total load in the network. On the other hand, they must also avoid overloaded nodes as they would become bottlenecks and would affect deeply the system efficiency (for example using a flow-control mechanism).

- As these solutions are aimed to P2P systems, they must not require the intervention of any central coordinator nor the use of global knowledge. The network itself, by the individual work of nodes, must be able to perform the adaptation pretended.

- An adaptative topology does not impose any forwarding politic, that is, the way walkers are routed through the network is independent of any change on the topology. Thus, adaptative systems can be combined with any of the systems presented in the previous section as well as others that the designers of the system think appropriate. As is explained in sections

3.3.1 and 3.3.2 below, in [54] and [14] a forwarding technique is suggested that directs queries to highly connected nodes.

It could be argued whether the term *unstructured* is well applied to adaptative systems, as there is a certain control on how the topology is built. But, as is stated in [54] "the topology adjustment, while it improves scalability, is not required for correctness of operation." It is not mandatory that the topology keeps certain properties to ensure a proper execution. Nonetheless, some topologies are more efficient than others. Adaptative topologies mechanisms try to form topologies that reduce the time needed to find resources.

DANTE, the P2P system introduced in this thesis, is based on the same idea and so implements a mechanism, executed by all peers, that adapts the topology to the load conditions. In this section we will introduce three other adaptative solutions previously presented to the research community.

3.3.1 Lv, Ratnasamy, and Shenker's Proposal

In [54] Lv *et al.* discuss that, although structured networks (which they advocate for) have better efficiency, maybe it is possible to find a solution to the lack of scalability of unstructured systems, and so making it possible to use unstructured networks when they are a better solution than structured ones (mainly, when the population is highly transient, there is natural replication of content and keyword searching is the common operation, see section 2.1.5).

Their idea is based on two previous works:

- In [5] (see section 3.2.2), Adamic *et al.* suggest to forward walkers to highly connected nodes in power-law topologies, although it does not propose any way to construct those topologies, nor does he take into account that high degree nodes could be overloaded by the extra work.

- Saroiu *et al.* in [68] study the characteristics of the Gnutella and Napster networks and show the heterogeneity present in those networks.

To make the adaptation, Lv *et al.* propose a distributed flow control and topology construction algorithm that avoids overloading nodes and makes the topology adapt, so high capacity nodes handle most queries that low capacity ones. Additionally, it is suggested to use a routing mechanism that forwards searches to high capacity nodes, which should be the ones with the highest degrees. This is similar to the proposal in [5].

The adaptation algorithm works as follows. Let c_i the capacity of node i, where c_i represents the number of messages that i is able to handle during a time interval T. Let $nbr(i)$ the set of nodes that are connected to i, its neighbors. For each node $j \in nbr(i)$, i keeps this information:

- $in[j,i]$: the number of incoming messages received by i from j during the latest time interval of length T. The total number of incoming messages into i is defined as $in[*,i] = \sum_{j \in nbr(i)} in[j,i]$, and is sent to all i's neighbors.

- $out[i,j]$: the number of outgoing messages sent to j by i during the latest time interval of length T.

- $outMax[i,j]$: the maximum number of messages that i can send to j per time interval T. Thus, $outMax[i,j] \geq out[i,j]$ at all times. Besides, being $(c_j - in[*,j])$ the spare capacity of j, then $outMax[i,j] - out[i,j] \leq (c_j - in[*,j])$. That is, i must avoid sending a number of messages to j beyond j's spare capacity. The purpose is to avoid overloading j.

Periodically, node i checks if it is overloaded. If so, it tells some neighbor j with high incoming rate $in[j,i]$ to change the connection between them to point to another neighbor p with high spare capacity. If i has no neighbor with available capacity, then it tells j to lower the amount of messages sent, that is, to lower $outMax[j,i]$. Note that peers need information about their neighbors' state. The authors state that each node i must communicate to each neighbor the total amount of incoming messages received $in[*,i]$.

At start time, peer i initializes the data about each neighbor j as follows:

- $in[j, i] = 0$
- $out[i, j] = 0$
- $outMax[i, j] = c_i/d_i$ where d_i is the degree of node i.

The initial state of a very simple network is depicted in Figure 3.2.

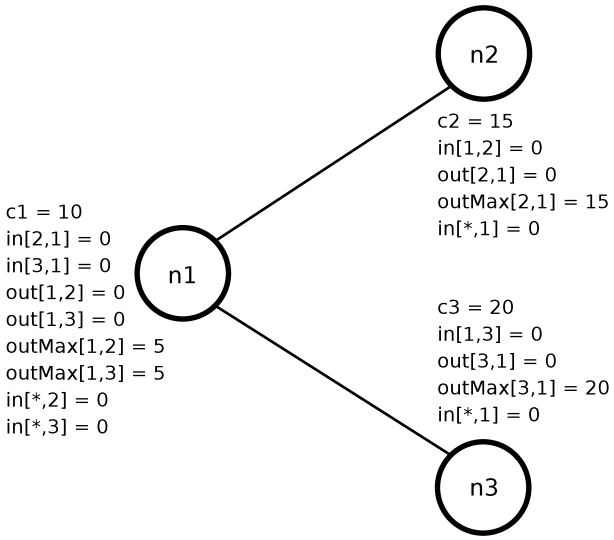


Figure 3.2: Lv et al.'s proposal of adaptative network, initial state.

To show how the adaptative algorithm works, we will use the network in Figure 3.2 and see what happens to node $n1$ when it becomes overloaded by $n2$'s queries, as is shown in Figure 3.3. Note that the in and out counters have changed, and $in[2, 1] = in[*, 1] > c_1$. This means that node $n1$ is overloaded, and it must tell node $n2$ to redirect its connection to $n3$, which has enough spare capacity. If node $n3$ had no spare capacity, then $n1$ would tell $n2$ to decrease $outMax[2, 1]$. In Figure 3.4 we see the resulting system (after the reconnection, the resulting counters are reset).

To evaluate the behavior of the system, the authors used simulations under two different scenarios. In the first one, the nodes capacities followed a Zipf-like distribution; in the second one, the distribution of nodes capacities follows

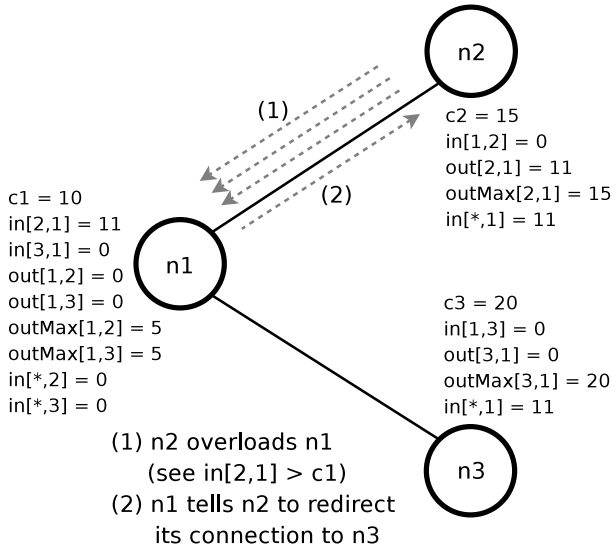


Figure 3.3: n2 saturates n1.

the distribution of nodes bandwidth capacities measured in [69]. In both cases the topology, starting from a random configuration, is adapted successfully. To observe how the topology evolves, the authors plot the average number of hops of searches, and how that value decreases as the simulation runs.

In these simulations the popularity of resources are set to follow a Zipf-like distribution, and the replication of each resource is proportional to its popularity. But a new idea is used: the number of resources held by each node is proportional to its capacity. As high capacity nodes tend to have more connections, they will be traversed by many searches. Besides, those nodes will know many resources in the system (as more powerful nodes store more resources). Thus, searches will be solved in few hops in many cases, as they will traverse high degree nodes which with high probability will know the resource looked for.

3.3.2 Gia

Gia, introduced in [14], is another proposal of an unstructured P2P network with a topology adaptation mechanism. In fact Gia is a refinement of the work

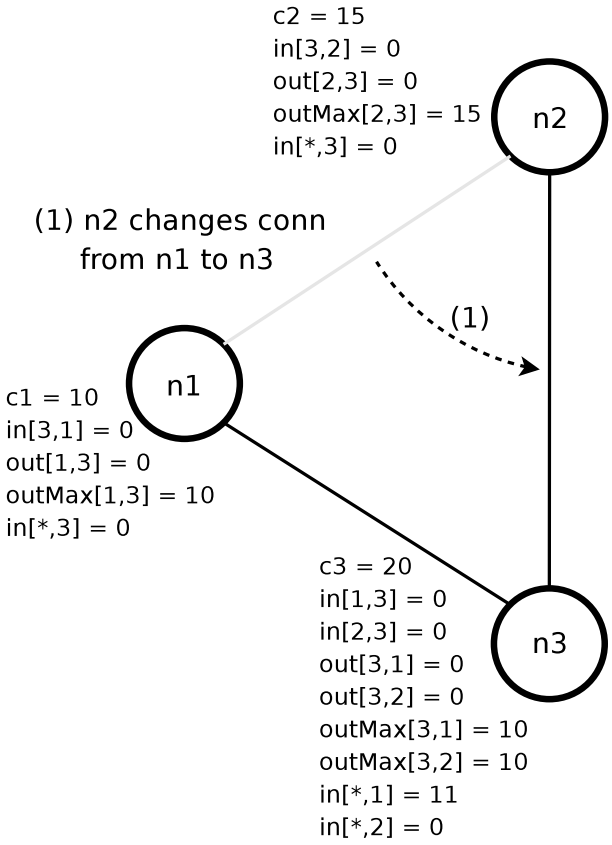


Figure 3.4: Resulting system after n2 redirects the connection to n3.

presented in the previous section, built by almost the same team of researchers. Thus it pursues the same goal of making high capacity peers to handle more messages than low capacity ones. To achieve this, most capable nodes must be very well connected so many walkers will reach them in few hops. Additionally, high capacity nodes must know a good amount of the resources in the system so they will be able to complete queries with high probability.

Gia has four key components:

- A *dynamic topology adaptation protocol*, which makes most capable nodes to establish more connections, and makes sure that well connected nodes actually have the capacity to handle all the incoming searches.

- An *active flow control scheme* that avoids overloading nodes. It assigns tokens to nodes depending on their capacity. Those tokens are then distributed among the peer's neighbors, each token representing a query that the neighbor is allowed to forward to the peer.

- *Proactive replication to neighbors*. Each node knows the resources stored by all its neighbors, so if the node receives a query for one of those resources it can reply on behalf of the neighbor that actually has the resource. Be aware that this ensures that high degree nodes will know many of the resources present in the system.

- A *modified random walk routing mechanism*, that forwards walkers to high capacity nodes.

Topology Adaptation in Gia

As said before, the goal of the topology adaptation functionality is to ensure that high capacity nodes are the ones with high degree. Besides, it guarantees that all peers are within short reach from one or more high capacity nodes. This section explains the fundamentals of the topology adaptation solution adopted by Gia.

In Gia's model, each node i has a capacity c_i that represents the number of messages per unit of time that i can handle. Additionally, each peer has two parameters that bounds the number of neighbors that node can have. These parameters are *max_nbrs* and *min_nbrs* which represent the maximum and minimum number of neighbors, respectively.

Periodically each node computes its *degree of satisfaction*, S, that represents how satisfied the peer is with its neighbors. When $S = 1$, the node is fully satisfied and so it does not need to keep changing its connections. But if $S < 1$ then the node must look for new neighbors to increase its S value. At all times, $0 \leq S \leq 1$. S is computed as follows (let d_i be the degree of i, and $nbr(i)$ the set of neighbors of i):

1. If $d_i \geq max\_nbrs$, then $S = 1$. Else, go to (2).

2. If $d_i < max\_nbrs$, then

$$S = \frac{\sum_{j \in nbr(i)} c_j/d_j}{c_i} \tag{3.1}$$

To explain Equation 3.1, first it must be noted that S represents how satisfied the peer is with its neighbors; more precisely, how satisfied it is with the capacity that those neighbors devote to the peer. While that capacity is less than the peer's own capacity, the node is not satisfied. In Equation 3.1 this is represented as follows. For each node j, c_j/d_j represents the share of j's capacity that is assigned to each of j's own neighbors. Then, $\sum_{j \in nbr(i)} c_j/d_j$ represents the total capacity that i can use from its neighbors, and thus dividing that value by c_i it a coefficient is obtained that represents how much capacity i is receiving from its neighbors versus the capacity that i is offering them. If the first value is lesser than the second, the node is unsatisfied (and the greater the difference, the lesser the satisfaction). The implicit idea is that one peer with capacity C will forward also approximately C queries, and so will need that neighbors offer him that same capacity.

Thus, nodes in Gia will look for new connections while their satisfaction is lesser than 1, as a new neighbor should increase its S value. The new neighbor to connect to is chosen from a local *cache* of known nodes (first a random subset of nodes is extracted from the cache, then the peer with the highest capacity inside the subset is selected). Once one candidate node is selected, the peer tries to establish a connection with it. Connections are established by a three-way handshake mechanism. As the candidate node's satisfaction can be affected by the new connection, it must decide if it accepts the node as a new neighbor or not.

Node i will accept immediately a connection from node j if $d_i < max\_nbrs$, as the new connection will always increase the satisfaction of i (see Equation 3.1). But in case that $d_i = max\_nbrs$, then i only can accept j's petition if it previously disconnects from some of its present neighbors. Thus, is must check if

there is some neighbor z suitable to be replaced by j. That neighbor must have a capacity lesser than c_j (so i's satisfaction is not decreased), and should not have less connections than j or at least should be well connected. The process to choose the neighbor z is as follows:

1. If $c_j > \max_{k \in nbr(i)}\{c_k\}$, then j is accepted and z is the neighbor such that $d_z = \max_{k \in nbr(i)}\{d_k\}$. Else, go to (2).

2. Let *subset* be the set of neighbors of i with a capacity lesser than the capacity of j, c_j. That is:

$$subset \ = \ \{k : k \in nbr(i) \ \wedge \ c_k < c_j\}. \tag{3.2}$$

   The candidate $z \in subset$ to disconnect from will be the one that fulfils these two conditions:

$$d_z \ = \ \max_{k \in subset}\{d_k\} \tag{3.3}$$

$$d_z \ > \ d_j + H \tag{3.4}$$

Condition in Equation 3.4 makes sure that poor-connected nodes are not dropped in favor of well-connected ones (H is an hysteresis factor). If finally i can not find a suitable neighbor z, then it rejects the connection petition from j.

Finally, the satisfaction level S of each node is also key to compute how often that peer must check if it must try to adapt its topology adding a new neighbor. The time between checks, I, is computed as:

$$I \ = \ \frac{T}{K^{1-S}} \tag{3.5}$$

where T is the maximum time interval, and K is a parameter that represents the *aggressiveness of adaptation*. It is straightforward from Equation 3.5 that satisfied nodes ($S = 1$) will only adapt their topology (if needed) every T units of time. But unsatisfied nodes will adapt their connections at a higher rate, the faster the lesser their satisfaction is, and/or the higher is the aggressiveness parameter.

Flow Control in Gia

The goal of the *flow control mechanism* of Gia is to avoid overloading nodes in the system. It is based in the concept of *tokens*. One token sent by some node i to any of its neighbors j represents a query that i is willing to accept from j. Then, j only will forward walkers to i if it has tokens from it.

Tokens are assigned at a certain rate, *token allocation rate*. The amount of tokens that node i sends to its neighbors depends on i's capacity c_i. If at some moment i receives more queries that it can process (for example, because it does not get enough tokens from its neighbors to forward the incoming search messages), it starts to queue them. If the queue gets too long, then it reduces the *token allocation rate* to reduce also the frequency of incoming queries.

To incentive nodes to announce their real capacity, each node i does not assign its tokens evenly. Instead, the number of tokens assigned to each neighbor j is proportional to the capacity of j, c_j.

Tokens are assigned using an algorithm based on *Start-time Fair Queuing* (SFQ) [33]. Each neighbor is assigned a fair-queuing weight equal to its capacity. Each time a new neighbor is connected, or some neighbor disconnects, the assignation of tokens is recomputed.

Replication of Neighbors Resources

In Gia, when two peers establish a connection they also exchange the indexes of the resources hold by each one of them. Hence, if nodes i and j are connected, and i receives a query for some resource stored in j, then i can reply that the resource was found successfully, as it knows where the resource is located. It is not needed that the query reaches j.

By this simple mechanism the efficiency of searches is highly increased, the main reason is that there will be well-connected nodes that will be able to reply many queries. Besides, those well-connected nodes, which are the ones that most likely will be able to reply queries, will be visited with high probability due to their high degree and the routing mechanism used in Gia that forwards

walkers to highly connected nodes (see next section).

Search Protocol in Gia

Searches in Gia have the following characteristics:

- Walkers do not follow a pure random walk. Instead, when forwarding a search, each peer selects the highest capacity neighbor for which it has tokens.

- Searches' scope is bound by a TTL mechanism and a *max_resp* parameter. Every time some walker finds a resource that matches the query, it sends a response message back to the source node that will follow the same path traversed by the query. It also decreases its *max_resp* counter. When a walker has performed TTL hops, or has found *max_resp* resources that matches the query, then the search is finished and the walker discarded.

- Nodes keep information about the searches (identified by an unique ID, if different walkers carry the same search, they have the same ID) they have processed previously, and to which nodes they were forwarded to. If some walker with an ID already processed visits the node, then the node will try to forward it to a different neighbor.

- When some node leaves the system, all the searches waiting in its queue are lost. To solve this problem, walkers send periodically *keep-alive* messages. The reply messages sent by walkers when they find some matching resource act too as implicit "keep-alives". If the walker has performed a certain number of hops without finding anything to report to the source, then it sends a dummy reply with 0 matching resources. If the source peer does not receive any reply nor keep-alive message for some time, it can re-issue again the same query.

Note that how searches are routed is not directly related with the adaptative nature of Gia. Another mechanism, like pure random walks, could be used as

well. Yet, the authors use this technique assuming that it will perform more efficiently in terms of searches duration.

Gia Simulations Results

The authors of Gia performed a series of simulations to study the system performance under different scenarios, and to compare it against other solutions like flooding.

As in [54], the nodes' capacities follow a distribution similar to the distribution of nodes' bandwidths measured for Gnutella (reported in [68]). The replication of resources is controlled by a *replication factor*, r, which states the fraction of nodes that store the resources looked for. That is, if $r = 1\%$, then only 1 node out of 100 holds the resource.

For these simulations, which were run with 10000 or 5000 nodes, max_nbrs was set to 128, and min_nbrs was set to 3. Additionally, to avoid that low capacity nodes split their capacity into too small parts, a new parameter was set, min_alloc. It represents the finest level of granularity a node capacity can be divided into. For each node i, it must be true at all moments that $c_i/d_i \geq min\_alloc$. Thus, a new condition must be accomplished when node i is deciding whether it accepts the connection petition from j:

$$min\_nbrs \leq d_i \leq min\{max\_nbrs, c_i/min\_alloc\} \tag{3.6}$$

To compare Gia with other solutions, the authors simulated also systems based on pure random walks and flooding over random topologies, and super-peers based networks.

To study Gia robustness when nodes fail (leave and enter the network), the authors also include some results obtained by simulating the performance of the system when nodes have a limited life-span. When a peer enters the network, it computes a time life t, where t is uniformly distributed between 0 and a parameter $max\_lifetime$. When t units of time have passed since the peer joined the network, then the node leaves the system. Immediately after that,

the node joins the system again connecting to a number of nodes chosen at random among the alive nodes in the network, and recomputes a new value for t. This aims to represent a constant churn of peers. To simulate higher churns, the value of *max_lifetime* was successively decreased in different simulations.

After analyzing the simulations results, the authors give the following conclusions:

- Gia outperforms all other systems, even by orders of magnitude, in terms of the query load it can sustain, or the number of messages required to find a resource.

- Setting *max_resp*= k (looking for k results) of some resource with replication r is equivalent to look for only one occurrence of some resource with replication r/k.

- Looking for high degree neighbors when forwarding walkers seems to have little impact on Gia's performance. This can be due to what happens at high loads: only high capacity nodes are able to handle queries, so walkers are anyway forwarded to the most capable peers (which will be the ones with the highest degrees) even when using pure random walks.

- Using replication of neighbors' resources has great importance.

- Heterogeneity allows to drop the average number of hops required by searches.

- Gia performs well even when there is a high churn of peers. Only under extremely high rates of nodes joining and leaving, the efficiency is affected.

The results of Gia are undoubtedly promising, and seem to confirm that using adaptative topologies can improve in a significant manner the performance of unstructured P2P systems. Yet, there are two circumstances that suggest that Gia results can be improved:

- Gia authors (apparently) do not know the work from Guimerà *et al.* [34] (explained in section 4.1), about optimal topologies in congested networks. They just follow some heuristics that seem to lead to certain topologies. Yet, by results in [34] we know that in certain scenarios other topologies will perform better (*e.g.*, a star-like topology is the optimal for non congested networks). DANTE, on the other hand, is based on the results obtained by Guimerà *et al.*, and in fact our simulations results, explained in Sec. 7.5.5 show how DANTE outperforms Gia in many scenarios.

- Gia nodes need to communicate often with their neighbors, to notify them their degree or the number of the tokens assigned to each one. This communication overhead affects the network results.

3.3.3 Cooper and Garcia-Molina Proposal

Cooper and Garcia-Molina present in [21] another system where nodes change their connections in order to form good topologies. Searches are performed by walkers following random walks. A particular characteristic of this proposal is that there are two types of links between peers (so in fact they build two different overlay networks):

- *Search links.* Links used to forward searches to nodes.

- *Index links.* Links used to send copies of content indexes between nodes.

Besides, links are unidirectional. So if there is a search link between peers A and B, $A \Rightarrow B$, then search messages can travel from A to B, but the contrary is not true (unless, of course, there is another search link from B to A, $A \Leftarrow B$). Likewise, and index link from A to B, $A \rightarrow B$ means that A sends its index of resources to B (so B knows A's resources), but the reverse is not true unless there is another index link from B to A, $A \leftarrow B$. If there are both an index update link and a search link from A to B we denote it as $A \Rrightarrow B$.

Indexes are not forwarded, so if $A \rightarrow B \rightarrow C$ then B knows A's and C knows B's indexes, but C does not know anything of A's resources. Both search and index update messages are assumed to load the nodes.

In their work, Cooper and Garcia-Molina propose and study different policies for *connect* and *disconnect* operations that allow networks to self-tune in order to improve efficiency (although, quite surprisingly, they do not propose any mechanism so nodes can decide to *which* other peers nodes should try to connect to). The topology can be adapted by nodes using two basic operations (*break* and *connect*). Like in the previous works, the basic idea is to prevent nodes from getting congested, in this case dropping connections using the *break* operation when needed.

The set of policies, represented by parameters, proposed by the authors is:

- Parameters for *connect* operations:

  1. Connect operations are *propertied* or not . A propertied connect tries to avoid *redundancy of coverage*. There is redundancy of coverage when one node A can reach another node B through more than one path.
  2. Connect operations are *two-ways* or not. A connect is said to be two-ways if it allows both nodes to search each other. That is, it is possible to find a resource in B arriving to A, and at the same time is possible to find a resource in A arriving to B. There are four possible two-ways connects:

     (a) I: $A \Leftrightarrow B$

     (b) II: $A \leftrightarrow B$

     (c) III: $A \Rrightarrow B$

     (d) IV: $A \Lleftarrow B$

     The type of two-way connect used is decided using probabilities: P_I, P_{II}, P_{III} and P_{IV}, where P_I denotes the probability of choosing a type I connect, and so on.

3. If the connect operation is not two-ways, there are two other probabilities to take into account:
   (a) Probability of building a search link instead of an index link: P_{sl}.
   (b) Probability of building a forward link instead of a backward link: P_f.

- Parameters for *disconnect* operations.
  1. *Break Threshold*, B_T, determines which load level (in number of messages received) is considered as congestion.
  2. *Policy to decide which connection to break:*
     (a) *MostLoadedLink*. Break the link that causes the most load, of those which load is above B_T.
     (b) *MostLoadedLinks*. Break all links whose load is above B_T.
     (c) *MostLoadedType*. If the load due to searches is greater than the load due to updates and greater than B_T, search links are broken. On the contrary, if load from update links is greater (and again above B_T), then update links are broken.
     (d) *MostLoadedLinkOfType*. Works like MostLoadedType, but instead of breaking all links of a certain type, it breaks only the most heavily loaded.
  3. *Break Interval*, B_I. Denotes the frequency at which load is checked, and break operations are performed if needed. If $B_I = 0$, then load is checked constantly.

Finally, the authors performed some simulations to test those different parameters and to compare the adaptative approach against superpeers based systems. In general, it can be concluded from their simulations that

- The consequences of each policy are strongly dependant on the scenario. Depending on the circumstances, one approach can have effects that are beneficial, detrimental, or even mixed.

- *Self-supervising* networks (as they call them) can outperform superpeer networks in many situations.

Chapter 4

Efficient Search Topologies

In section 3.3 we have described some proposals that try to improve the efficiency of random walks by adapting the topology to the load conditions, where the term *efficiency* refers to the average time that it takes to solve searches. All those solutions are oriented to avoid overloading nodes by means of negotiation processes.

Yet, none of those works try to characterize which are the topologies that minimize the search time in the scenario of P2P networks. Thus, a question that can be immediately raised is whether the topologies formed by those systems are in fact optimal, or better ones can be achieved maybe using different approaches.

In this section is introduced the work of Guimerà *et al.* [34] about optimal topologies, that is an initial attempt to find out which are the best network configurations for searching, assuming the search process is Markovian (the next hop does not depend on the previous ones). It turns out from Guimerà *et al.*'s results that the topologies achieved by Gia and other works are far from being the optimal ones in many situations.

Along with the research of Guimerà *et al.*, this section also explains the work of Cholvi *et al.* [16, 17], which represents a first attempt of developing an adaptation mechanism that:

- Is suitable for P2P networks (no coordinator is needed, nor global knowledge is assumed).

- Leads the network to configurations close to those that Guimerà *et al.* estimate as optimal.

4.1 Optimal Search Topologies

An important issue that has raised great interest in the research community on P2P systems and overlay networks in the latest years is the identification of the overlay network topologies that, given a search mechanism and a traffic pattern, minimize the average time needed to perform a search. Those are called *optimal overlay topologies* and, clearly, should be the topologies of choice in practical overlay networks. However, optimal topologies vary with the system conditions. Therefore, a natural guide to drive the evolution of dynamic overlay networks consists of obtaining, at each precise moment, the corresponding optimal topology.

Regarding the structure such "optimal" networks should have, Guimerà *et al.* [34] have reported some interesting results. Namely, they characterized which are the optimal topologies in a scenario where the search mechanism uses a shortest walk and each node has information about the resources held by its first-order neighbors. Three facts can be concluded from this work:

- *When the system is not congested the optimal topology is a star-like structure formed by a small number of central nodes with the rest of nodes connected to them.* It is straightforward that in this scenario, all queries are solved in, at most, one hop, as central nodes know all the resources in the system, and all walkers will always reach a central node at first hop as all connections point to them.

- *When the system is congested, the optimal topology is a random-like one*, where all nodes have approximately the same degree.

- *There is a sharp transition between these two optimal topologies*, contrary to the natural hypothesis of having a wide range of optimal structures in between.

Although initially surprising, it is not hard to understand the reasons of the third fact. First, we must be aware that the load on the system depends strongly on the average number of hops taken by each search, the *average search length*. If the average search length is increased, then the load in the system is also increased. Now, let's assume we start from a centralized topology, and the load is such that some central nodes get congested (their processing rate is lesser than the search arrival rate). To avoid this congestion, some of the connections to those central nodes are redirected to other nodes in the network (not central). Initially, it could be expected that congested nodes can again process all incoming searches. But, in fact, since the topology is now more 'random', searches need more hops to complete, and so the load in the system raises. Thus, again nodes can become congested, and so connections should must be changed again to a more random topology. If this process goes on for a few steps, finally the network will reach a totally randomized state. It is easy to realize that the range of loads that allow to find a balance between random and centralized topologies is thus quite narrow.

In general, Guimerà *et al.* results are very interesting and could be used in almost any system where its entities organize as a network. However, the authors did not say how these topologies could be achieved dynamically in a real system. In real P2P systems, we face two problems when using their approach:

- First, the simplest ways to identify the optimal topology requires a global knowledge of the network, which is usually not available in a real P2P system.

- Second, once the desired topology is identified, the simplest ways to achieve it is to use a global mechanism to coordinate how nodes have to establish connections.

Thus, applying the conclusions of this work to P2P networks is not straightforward. A topology adaptation mechanism that fits P2P systems should be run locally at the nodes, and should not need global knowledge.

Besides, it is worth to note that nodes in the networks studied by Guimerà *et al.* are *homogeneous*. Real networks, nonetheless, are known to be *heterogeneous* [68], and in that scenario some of the results of this work could not hold.

It is interesting to recall a work from Paul Silvey and Laurie Hurwitz [70], where authors try to find efficient topologies for resource location when using flooding. In order to get better topologies, two operations are performed (global knowledge is used):

- Removal of redundant links.
- Connecting nodes whose minimum path length is the diameter of the network (so the average distance between nodes is decreased).

A very interesting result is that as the topology evolved using those two operations, hubs started to appear naturally, although the authors do not pursue them. As they state in their conclusions:

> "...hubs happen, and unless you try to prevent them, they seem to be nature's way of finding good solutions to many network problems."

Although starlike topologies are not mentioned in their work, it seems straightforward to relate this conclusion with the fact that centralized networks are efficient (and often optimal). In general, we can infer that to improve searches efficiency we must pursue strongly clustered topologies, being the centralized topology the extreme case.

4.2 Building Optimal Topologies

As mentioned before, obtaining the optimal topologies proposed by Guimerà *et al.* is not an immediate task in pure P2P networks, where no central coordinator nor global knowledge are available. To apply Guimerà *et al.* results to P2P systems, we must find a topology adaptation mechanism that:

- Works using local knowledge: nodes only need to know their state (although nodes also know their neighbors' resources, to be able to reply in their behalf).

- It is run by all nodes, that is, all nodes participate in the topology adaptation by executing the same functionality (there are not different roles).

- Makes the network evolve to a centralized topology when the load is low, and to a random topology when the load is high.

In a first attempt to face these issues, Cholvi *et al.* [16, 17] proposed an innovative solution, where nodes establish links to other peers using an *attachment kernel* function Π_i, that denotes the probability of connecting from any node to peer i (the greater Π_i is, the more 'attractive' that node is). It has the form

$$\Pi_i = k_i^{\gamma_i}, \tag{4.1}$$

where k_i denotes the degree of node i and γ_i is defined as

$$\gamma_i = \begin{cases} 2 \text{ if } c_i < \mathit{threshold} \\ 0 \text{ otherwise.} \end{cases} \tag{4.2}$$

c_i represents the load on node i, and *threshold* represents the amount of load the peer can process without considering it congested.

The form of γ_i can be reasoned as follows. First, if γ_i is set to 0 for all nodes, a random topology is built [46], as Π_i would then be 1 for all peers, and so all peers are equally likely to be chosen as new neighbors to connect to. That will lead to a topology where all nodes have a similar degree. In turn, setting γ_i to 2 for all nodes, the topology obtained is centralized [46]. Because of the power form of Π, a little difference in the degrees of two nodes would make much more attractive the node with highest degree, and so more peers will connect to it increasing again its Π value.

Hence, if nodes do not get overloaded ($c_i <$ threshold and so $\gamma_i = 2$), the network will tend to form centralized topologies. But if the load grows and nodes get congested, then the topology will evolve to a random one.

In [16, 17] the authors execute some simulations to study the behavior of networks with this adaptation mechanism. In those simulations the load on each node i is estimated using the *betweenness centrality* of the node (defined as the number of routes, through shortest paths, that cross it). The goal was to minimize the global load on the network, since, by Little's Law, minimizing it is equivalent to minimizing the cost of search. From those simulations the authors conclude that the attachment kernel successfully adapts the topology, from centralized to random as the load is increased. Besides, they also show how, as predicted by Guimerà, the centralized topology is optimal for low loads, and the random one for high loads.

The work of Cholvi *et al.* is very interesting and valuable as a first proposal that introduces innovative ideas to increase P2P systems efficiency through network self-adaptation. But there are some issues related with their experiments configuration that raise concerns about the validity of their conclusions in real networks. Mainly, those concerns are:

- The simulations run did not use packet injection to simulate load. Instead, load was estimated, as described above, using the nodes' centrality.

- Centrality was estimated using shortest paths, yet in real scenarios searches will use random-walks.

- In this simulations, it is not defined how nodes can gather information from other peers. Several techniques could be applied but it is unknown how they will affect the system performance, and if the data obtained will be enough for an effective topology adaptation.

In chapter 6 we present an experimental implementation of this proposal on a real network that validates main of Cholvi *et al.*'s conclusions. Yet, that imple-

mentation also allowed us to detect some problems related with the attachment kernel:

- It is necessary to find a threshold that truly represents each node's capacity. In fact, simulations show, by setting different thresholds, that this parameter has a strong incidence on system efficiency.
- No consideration about network heterogeneity is taken into account.
- Sharp changes on the Π_i value can happen when nodes with high degree (and so highly attractive) become congested. In that case, the Π_i value falls to 1, making neighbors to disconnect quickly from the node. This will make the node unable to reply many of the searches in its queue (remember that each node knows only the resources of its neighbors), so those searches will have to be forwarded again. If this process of nodes attracting many connections (and hence many walkers) quickly and then sharply getting overloaded (and so making neighbors to disconnect from it) repeats frequently, then the impact on the performance can be high.

In chapter 7 we introduce DANTE, our P2P system that modifies and improves Cholvi *et al.*'s proposal, aiming to solve those issues.

Chapter 5

An Analysis of Random Walks in P2P Networks

In this chapter we obtain a set of equations that capture the behavior of random walks in unstructured P2P networks where each peer knows its neighbors' resources (one-hop replication), so it can reply on their behalf. This is a typical assumption in previous works on dynamic topologies, and also a characteristic of real world networks that has been taken into account in other research works that try to model random walks [5].

The goal of our model is to offer good estimations of some important system metrics. The analysis has two parts. The first one computes the average number of nodes known by the random walk at each hop l (some authors call this the portion of the network *covered* [5]), and the average number of hops $\overline{l}$ that a query message takes before finding a resource. The second part uses $\overline{l}$ to estimate the average search time (time for searches to be solved), applying queuing theory assumptions and concepts.

5.1 Previous Work About Random Walks

There are several research works [59] [5] that have already tried to describe the behavior of searches by random walks. This interest raises from the fact that, in many real networks, participants do not have information about where the

resource they look for can be located. Thus, they can not use heuristics that help to guide the search through the network. As a result, random walks become a feasible option to route searches.

The main difficulty found when trying to model random walks is the fact that they can not be assumed to be a stochastic process without state in many real world networks. This is specially true when the degree distribution is highly skewed, so a few peers have a degree much higher than the rest, like in power-law networks. The problem is that in those networks the random walk tends to 'gravitate' around high degree nodes. Thus, the probability of success is initially higher, as well connected nodes know much about the network. Hence, the random walk has more chances of success when visiting those nodes. But, as more hops are taken, the probability that each hop just revisits the same high-degree peers again and again grows, and so the probability of success decreases. Besides, and even if the node visited at a certain hop was not traversed before, many of its k links will point to already known peers, so the success probability can not be approached by k/N. In other words, random walks can not be directly modeled as a typical *balls and bins* problem.

This is confirmed by Adamic *et al.* in [5]. There, they propose a set of expressions to characterize random walks in power-law networks, using the generating function formalism [84] introduced by Newman [59]. When trying to analyze the expected average search length, they found an important divergence between their predictions and the results obtained by simulation. The same happened for their analysis of the expected cover time (number of hops required to know a part of the network). They also point out the reason:

> "...for the power-law graph, when 50% of the graph has been visited, nodes are revisited about 80% of the time, which implies that the same high degree nodes are being revisited before new low degree ones. It is this bias that accounts for the discrepancy between the analytic scaling and the simulated results..."

Those results are specially interesting, as many important networks that

are built in a decentralized manner, like the Internet [23, 27] or some P2P networks for the sharing of media content [40], seem to have power-law degree distributions.

In a previous work, Newman *et al.* [59] proposed an expression to compute the average shortest path length between two nodes on a network with an arbitrary degree, but they did not include experimental results for that magnitude, and they do not take into account the problems described above. Newman's work is nonetheless interesting, as it studies a variety of properties (*e.g.* the giant component size) of random graphs with arbitrary degree distribution. The importance of this work comes from the fact that the authors do not assume the degree distribution to be a Poisson distribution (this assumption, for example, is used for epidemiological models [47]). Nonetheless, other works had already shown that in many scenarios the distribution of vertex degrees does not follow a Poisson. Some examples cited by the authors of [59] are the network of telephone calls [4, 6] or the power grid [74].

The model presented in this thesis also has the same goal of modeling networks of any degree distribution, although it takes a different approach to the problem and is focused on the behavior of random walks on these networks, instead of other measurable properties.

Finally, Gkantsidis *et al.* [31] argue that a random-walk can simulate independent uniform sampling of peers in a P2P network. Using [31] as a basis, Bisnik *et al.* [10] propose a expression for the average search length in a P2P network *without* one-hop replication. They state that, if a resource has a replication of $p = r/N$ (where r is the number of nodes that have the resource in a network of N peers), the probability that the resource is not found in T steps is $(1-p)^T$. Their reasoning is that a random walk of T hops has the same properties of T independent samples. That is, they assume that the random walk can be modeled like a balls and bins problem. In our opinion, unfortunately, this is a *misinterpretation* of [31]. Truly enough, their simulations seem to support their model, at least to a certain extent, but we deem this is due mainly to the

small maximum value set for T (100 hops, 1/100 of the size of the network).

In the remaining of this chapter we draw our own proposal of a model that tries to reproduce the characteristics of random walks in P2P systems given the overlay network distribution.

5.2 Estimations of Known Nodes and Searches Length

As explained before, we aim to find an analytical expression that computes the average search length in number of hops in a P2P network. That is, the average number of hops, $\overline{l}$, that a random search, started from any node a, needs to take for finding any other node d, given the degree distribution. We assume that each node knows its neighbors, so in fact we compute the number of hops to find some neighbor of d from a. This is equivalent to perform a search for a resource located on node d in a network where each node knows the resources held by its neighbors (one-hop replication).

Unlike other works, this proposal does not define random walks as a stateless process. Instead, random walks are associated to a sequence of *states*. Each state of the random walk depends on the number of hops performed. This state is defined using a set of metrics that estimate the increment of *knowledge* about the network achieved by the random walk at each hop. These metrics take into account the probability that the node visited at each hop was traversed before, or the probability that its links point to already known nodes.

The state of the random walk allows us to compute the success probability at each hop. Using this probability, we can then estimate the expected average search length.

We assume that, for each network, we only known its degree distribution. Along with our equations we present and discuss the results of some experiments that confirm the validity of our model.

5.2.1 Conventions

Let G be a graph with N nodes. The degree distribution is given by p_k, where p_k is the probability that some node of the network chosen uniformly at random has degree k. n_k denotes the number of nodes that have degree k ($\sum_k n_k = N$, $\forall k\, p_k = n_k/N$). The average degree is given by $\overline{k} = \sum_k k p_k$. We define *endpoint* as each one of the two ends of a link between two nodes. We denote by S the total number of endpoints in the system $S = \sum_k k n_k = N\overline{k}$.

We will study first how to obtain, hop by hop, the following values:

- V_k^l Average number of *different* nodes of degree k *visited* by the random walk until hop l (including the peer visited at that hop).

- L^l Average number of *different* nodes' endpoints that have been checked until hop l, included. If we arrive to a node of degree k, and that node has not been traversed before, then we are checking k connections. See Fig. 5.1.

- C_k^l Average number of *different* nodes of degree k that have been *checked* by the random walk until hop l, included. A node is checked if the random walk visits the node or one of its neighbors. We also call them *known* nodes. It can be viewed as a metric of how much of the network has been *covered*.

All these values help to model the behavior of the random walk. Finally, they will be used to compute $\overline{l}$, the estimated average searches length.

5.2.2 Assumptions

All nodes are equally likely to be the starting point or target of the random walk. The network has no loops (there are no links connecting a node to itself) nor multilinks (there can not be more than one node connecting two nodes).

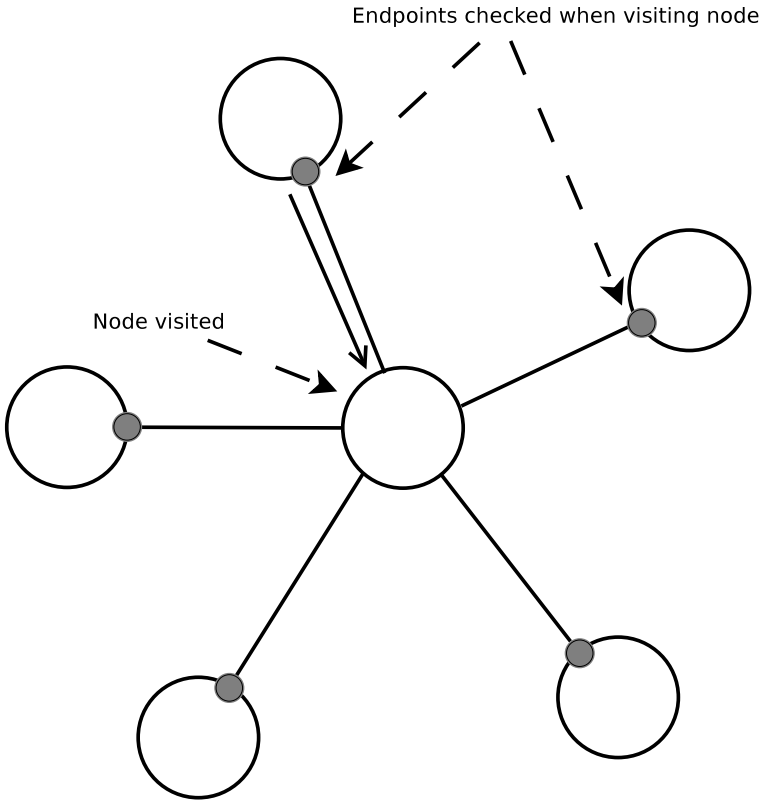


Figure 5.1: Endpoints checked when visiting a node.

The *probability to arrive to some node of degree k at a certain hop,* $P_A(k)$ is proportional to k, and is computed by the following expression [5]:

$$P_A(k) \;=\; \frac{kn_k}{S} \;=\; \frac{kp_kN}{\sum_j jp_jN} \;=\; \frac{kp_k}{\sum_j jp_j} \;=\; \frac{kp_k}{\overline{k}} \tag{5.1}$$

Besides, *all nodes with the same degree have the same probability of being visited by the random walk*. That is, if the random walk visits a node of degree k, then the probability of that node to be any of the n_k nodes with degree k is $1/n_k$. So the probability of visiting a specific peer at each hop, if that peer has degree k, is $P_A(k)/n_k$.

5.2.3 Visited Nodes

As said before, V_k^l is the number of different nodes of each degree k that have been visited by the random walk after l hops.

All nodes in the network have the same probability of being the search origin, so the probability of starting a search from a node of degree k is $\frac{n_k}{N} = p_k$. We denote by $l = 0$ the initial state of the walk, and we estimate V_k^0 as:

$$V_k^0 \;=\; 1 \cdot p_k \;+\; 0 \cdot (1 - p_k) \;=\; p_k \tag{5.2}$$

For the first hop $l = 1$, we have that:

$$V_k^1 \;=\; V_k^0 \;+\; (1 \cdot P_A(k) \;+\; 0 \cdot (1 - P_A(k))) \;=\; V_k^0 \;+\; P_A(k) \tag{5.3}$$

After the first hop, for $l > 1$ we must take into account the probability that the node arrives to a node already visited. To compute that probability, we define these two values:

- $P_{\text{Visited}}(k, l)$. It represents the probability that, if the random walk arrives to a node of degree k at hop l, that node has been visited before. Recall that the number of nodes of degree k is n_k, and the number of nodes of that degree visited during the previous $l-1$ hops is given by V_k^{l-1}. Thus, a first approach to compute the probability of arriving to an already visited

node, if that node has degree k, would be V_k^{l-1}/n_k. However, V_k^{l-2} is more accurate, as the node the walk visited at hop $l-1$ can not be visited at hop l (there are not loops in the network), and so it should not be taken into account. Thus, finally, the probability that the node has been visited before is computed as follows:

$$P_{\text{Visited}}(k,l) \;=\; \frac{V_k^{l-2}}{n_k} \tag{5.4}$$

- P_{Back}. It is the probability that at hop l the random walk is moving backwards, returning to the node where it came from at hop $l-1$. The node visited at hop $l-1$ has degree j with probability $P_A(j)$. In that case, the random walk will go back through the same link it came with probability $1/j$. So we define P_{Back} as:

$$P_{Back} \;=\; \sum_j P_A(j)\frac{1}{j} \;=\; \frac{1}{\overline{k}} \tag{5.5}$$

Using these probabilities we define V_k^l when $l > 1$ with the following expression:

$$V_k^l \;=\; V_k^{l-1} + P_A(k)(1 - P_{\text{Back}})(1 - P_{\text{Visited}}) \tag{5.6}$$

Expanding this expression we obtain that:

$$V_k^l \;=\; V_k^{l-1} + \frac{kp_k}{\overline{k}}\left(1 - \frac{1}{\overline{k}}\right)\left(1 - \frac{V_k^{l-2}}{n_k}\right) \tag{5.7}$$

In the following figures we compare the results obtained experimentally and the estimations obtained with 5.7. Three experiments were run, each with different networks types:

- An *Erdos-Renyi* network [12], with an average degree $\overline{k} = 10$. To build an Erdos-Renyi network, all pairs of nodes of the network are considered, and a link is established between them with a probability p (to get an average degree of 10, we set $p = 10/N$).

- A power-law network, built using the mechanism proposed by Barabasi-Albert in [9]. This algorithm starts with a small number of initial nodes n_0, $n_0 << N$. Then, at each step, a new node is added to the network. This node is then connected to some of the already present nodes, choosing its new neighbors in a *preferential* way, *i.e.* the probability of connecting to some node i is proportional to its degree k_i. That is, if we denote pl_i as the probability of building a connection with node i, then $\mathrm{pl}_i = k_i / \sum_j k_j$. For our networks, we used an initial number of fully connected nodes $n_0 = 5$, and for each new node built 5 new connections. The resulting networks always had an average degree of $\overline{k} = 10$.

- A power-law network, built in two steps. First, the Barabasi-Albert method described above was used (with the same parameters than before) to obtain a network with a power-law degree distribution and an average degree of $\overline{k} = 10$. From that network we extracted the degree distribution and applied the algorithm proposed by Newman to build networks of arbitrary degree [59]. This mechanism is simple: first, no connection from any node is linked. Then, iteratively, pairs of free connections are chosen uniformly at random and linked. We have slightly modified the algorithm to avoid loops and multilinks.

Our experiments have a strong focus on power-law networks for two reasons. First, some real-world networks are known to follow a power-law distribution degree, like the Internet [27, 45, 7] or Gnutella [40, 65], and so these topologies have gained special attention from the research community before [5, 9]. Second, we are interested on testing our model with topologies where the degree distribution is skewed. In these networks random walks will have the behavior described in Sec. 5.1: the tendency to revisit certain nodes due to their high degrees. This is accomplished by power-law networks.

The motivation for running experiments with power-law networks built by two different mechanisms is that the Newman algorithm can build, with identical probability, all the possible networks that have the given degree distribution.

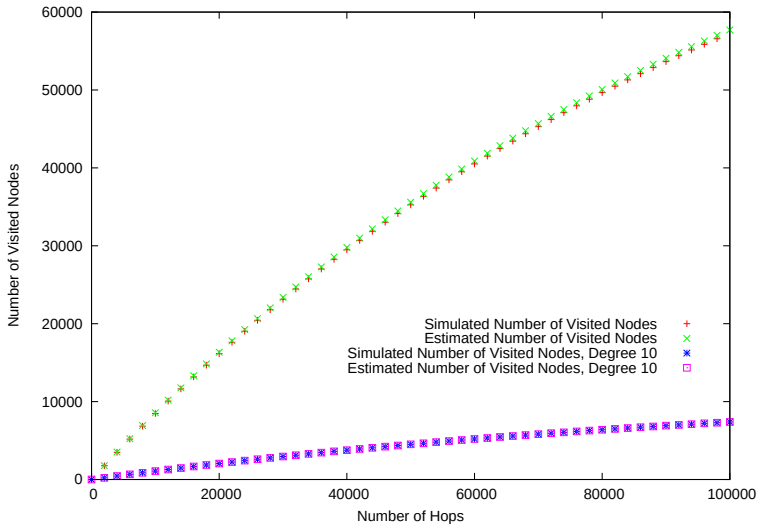


Figure 5.2: Visited nodes experiments and estimations results, Erdos-Renyi network.

This is not ensured by the Barabasi-Albert method. Note that, as the model takes as input the degree distribution, it will predict similar values for the two kinds of networks.

All experiments were run with 10^5 nodes. In each figure we show how the number of different traversed nodes evolves as hops are performed. We focus on two parameters in particular: the number of *total* different nodes visited at hop l of all degrees (*i.e.* $\sum_k V_k^l$), and the number of different visited nodes of degree $\overline{k} = 10$ at hop l (*i.e.* V_{10}^l). We also show the results predicted by the model, given the distribution degree of the network. The experimental results were obtained running 10^4 random walks over each network. All random walks had a length of 10^5 hops.

Results are shown in Figs. 5.2, 5.3 and 5.4 (for the sake of clarity, we plot the results every 2000 hops). By the results we can conclude that the predicted values are close to those obtained by the experiments, for all networks.

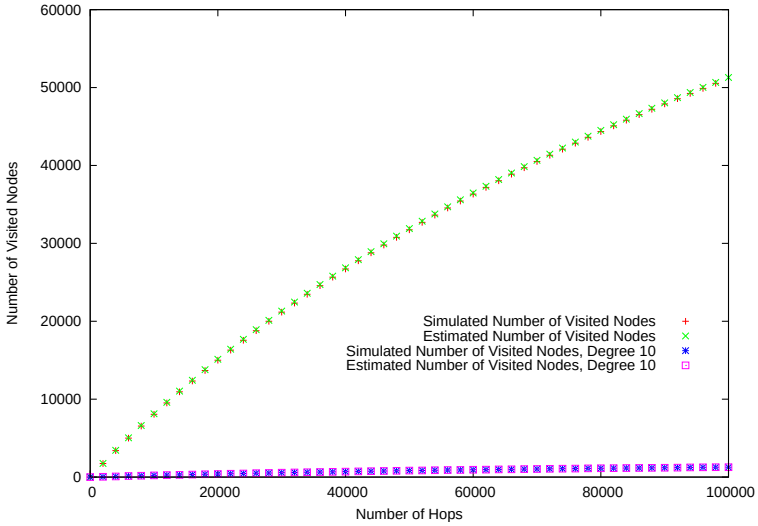


Figure 5.3: Visited nodes experiments and estimations results, Barabasi-Albert network.

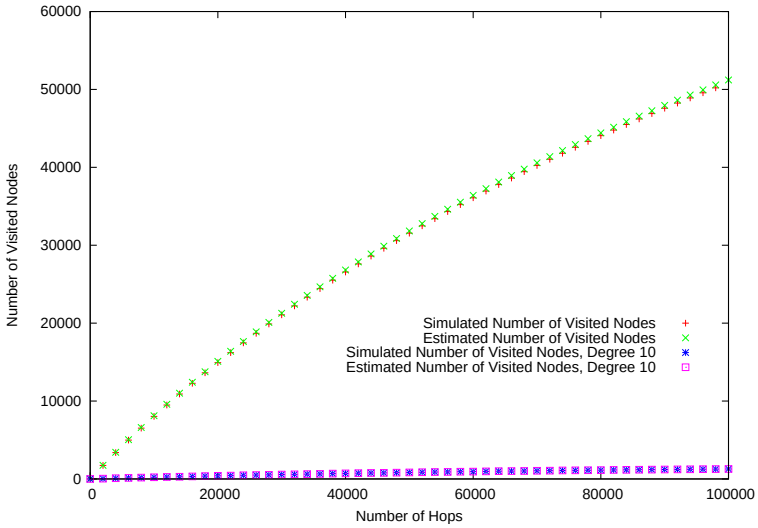


Figure 5.4: Visited nodes experiments and estimations results, Newman network.

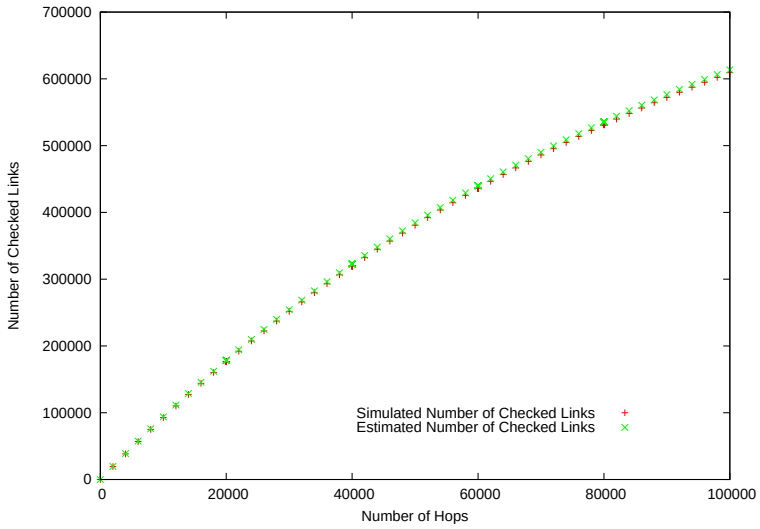


Figure 5.5: Checked endpoints experiments and estimations results, Erdos-Renyi network.

5.2.4 Checked Endpoints

This metric represents the average amount of different endpoints checked after hop l, and we denote it L^l. We define *checked endpoint* as the endpoint that is at the other end of each one of the links of a visited node (the endpoint on the neighbor's side). See Fig. 5.1 for an example. L^l is easily computed using the V_K^l values:

$$L^l = \sum_k k V_k^l \tag{5.8}$$

Three simulations (see Figs. 5.2, 5.3 and 5.4) were run to check the goodness of this model, for the same three type of networks than before, and also with the same parameters. The experiments show that the results predicted by the model are a good approximation of the values obtained experimentally.

5.2.5 Known Nodes

This metric is an estimation of the average number of different nodes known after hop l, C^l. Recall that a node is known if the random walk has visited it

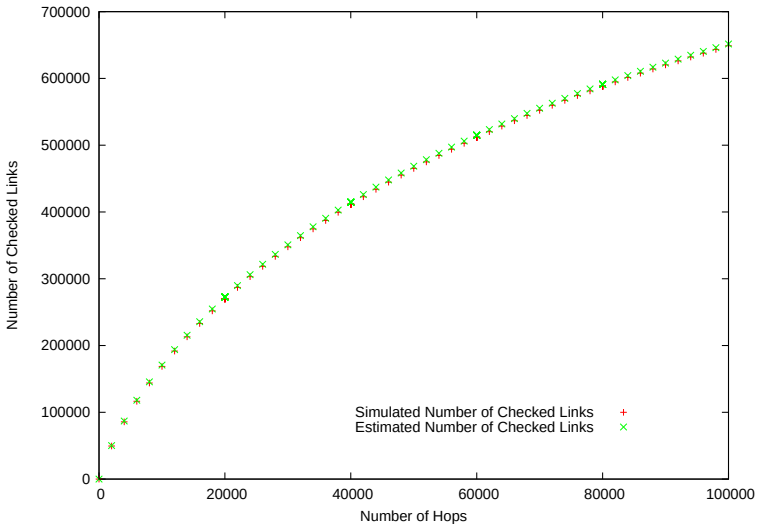


Figure 5.6: Checked endpoints experiments and estimations results, Barabasi-Albert network.

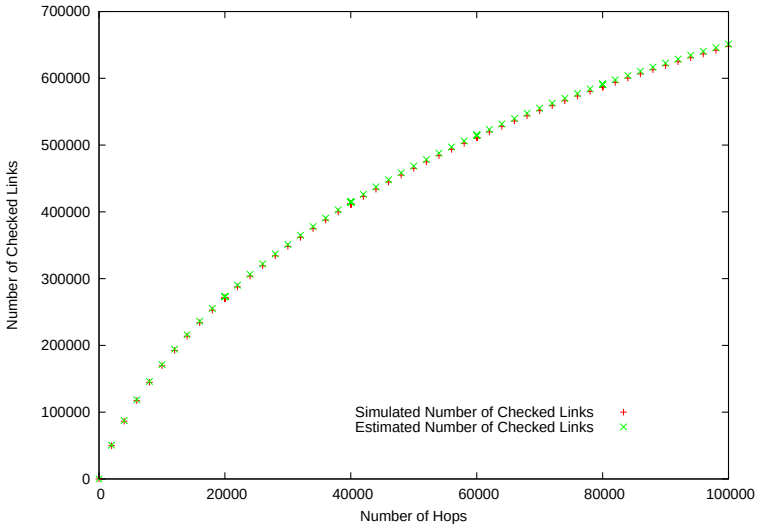


Figure 5.7: Checked endpoints Experiments and estimations results, Newman network.

or any of its neighbors. To obtain its value, first we will compute the number of different nodes of each degree k known at that same hop, C_k^l.

For hop $l = 0$, we have that:

$$\begin{aligned} C_k^0 &= p_k(1 + kP_A(k)) + \sum_{j \neq k} p_j j P_A(k) \\ &= V_k^0 + \sum_j p_j j P_A(k) \end{aligned} \tag{5.9}$$

The intuition behind this equation is as follows. The first term represents the value to add to C_k^0 to take into account the possibility that the initial node has degree k. The second term refers to the number of endpoints of the initial node that point to nodes of degree k. If the source node has degree j (which will happen with probability p_j) then, in average, $jP_A(k)$ nodes of degree k will be known, as each one of the j endpoints of the source node will point to a node of degree k with probability $P_A(k)$.

Now, for $l > 0$, $V_j^l - V_j^{l-1}$ represents the number of nodes of degree j that are visited for the first time at hop l. We are interested only in those nodes, as arriving to an already visited node would not increase the C_k^l values.

With this in mind, we want to compute the number of endpoints that will be checked for the first time at hop l, and thus can belong to nodes not known yet. This value can be regarded as the number of *trials* at hop l. We denote this magnitude Tr^l. A first approach could be to compute it simply as $\mathrm{Tr}^l = L^l - L^{l-1}$. However, we must realize that one of the endpoints checked at hop l points to the node the random walk comes from. Thus, that endpoint can neither lead to the destination node, nor increase the number of known nodes, and so it should not be counted in Tr^l. This is graphically shown in Fig. 5.8. Again, we draw in grey the endpoints checked at hop l. The number of grey endpoints is thus given by $(V_k^l - V_k^{l-1})k$. But note that one of those endpoints leads to the node visited at the previous hop, and so it should not be taken into account to compute the number of *trials*. Hence, a more accurate definition of Tr^l is given by:

$$\mathrm{Tr}^l = \sum_k (V_k^l - V_k^{l-1})\,(k-1) \tag{5.10}$$

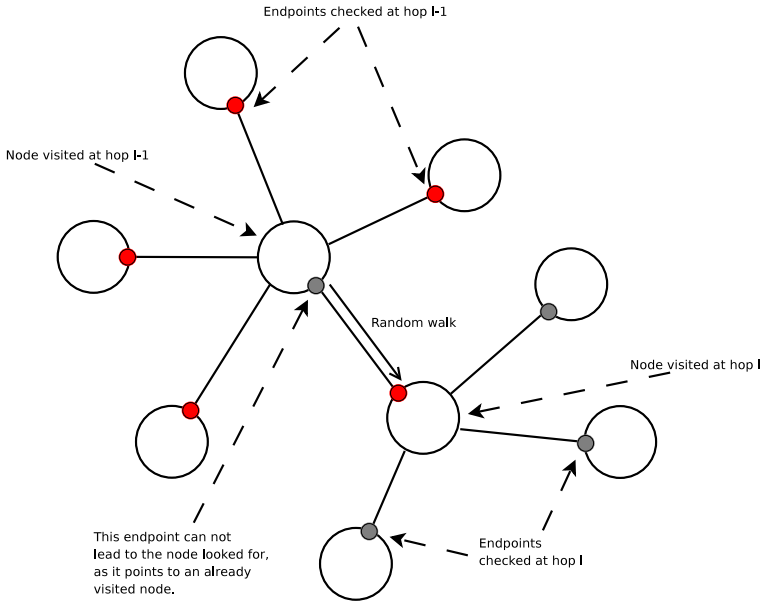


Figure 5.8: Connections checked when visiting a node.

Finally, we need to compute the probability that each link of the peer visited at hop l points to a node that has not been known before (recall that we are assuming that the node is visited for the first time, otherwise that probability would be 0). This is computed as:

$$\frac{k(n_k - C_k^{l-1})}{S - L^{l-1}} \tag{5.11}$$

This expression is explained as follows. The sum of all endpoints of all unknown nodes with degree k is given by $k(n_k - C_K^{l-1})$. On the other hand, the number of endpoints that each link can point to is $S - L^{l-1}$. S is the total amount of endpoints in the network, L^{l-1} is the number of endpoints already checked by the random walk. We subtract L^{l-1} from S because the node the random walks arrives to at hop l was not visited before, and so none of its links can point to endpoints that were already checked. By dividing these quantities we get an approach to the probability looked for: the probability that each one of the endpoints of the visited node points to an unknown node of degree k.

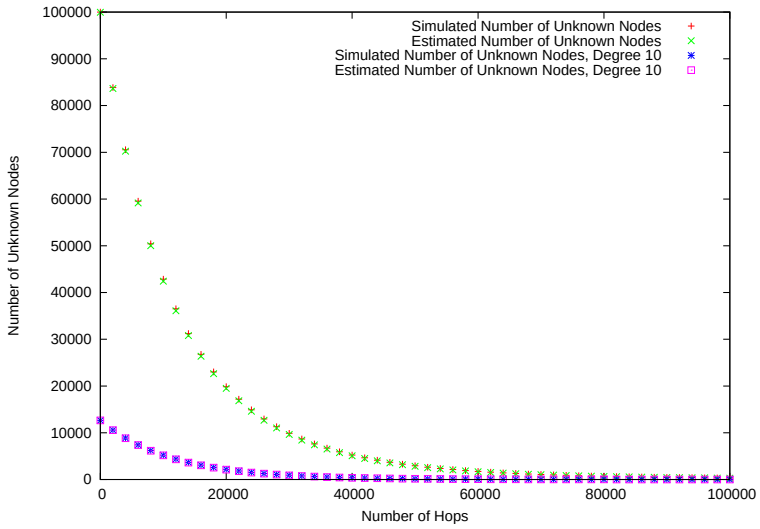


Figure 5.9: Known nodes experiments and estimations results, Erdos-Renyi network.

With this probability, we can compute C_k^l as:

$$C_k^l \;=\; C_k^{l-1} \;+\; \left(\frac{k(n_k - C_k^{l-1})}{S - L^{l-1}}\right) \mathrm{Tr}^l \tag{5.12}$$

Finally, the total number of nodes known at hop l is computed by:

$$C^l \;=\; \sum_k C_k^l \tag{5.13}$$

From this, the fraction of the network covered can be obtained trivially by C^l/N.

We have tested this model using, as before, three different network models: Erdos-Renyi, Barabasi and Newman. We focus on two values, the total number of *unknown* nodes, $N - C^l$, and the number of unknown nodes of degree $k = 10$, $n_{10} - C_{10}^l$, at each hop l. Again, the experimental results are computed as the average of the results obtained from running 10^4 random walks for 10^5 hops.

The results are shown in Figs 5.9, 5.11 and 5.13. The results for the same experiments, but in logarithmic scale, are shown in Figs 5.10, 5.12 and 5.14.

The values predicted by the model remain very close to the values obtained experimentally for the Erdos-Renyi and Newman networks, but there is a certain

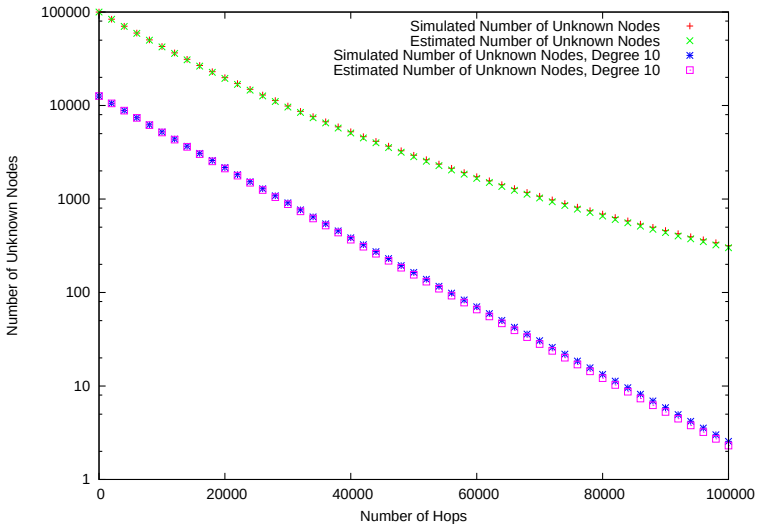


Figure 5.10: Known nodes experiments and estimations results, log scale, Erdos-Renyi network.

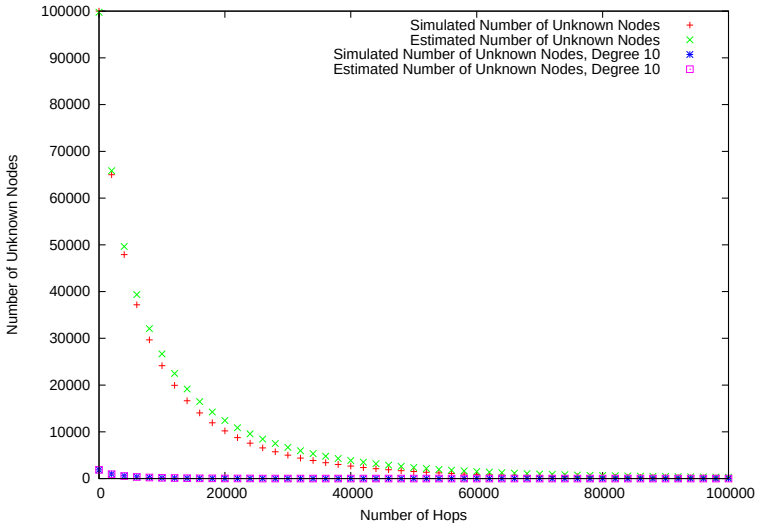


Figure 5.11: Known nodes experiments and estimations results, Barabasi-Albert network.

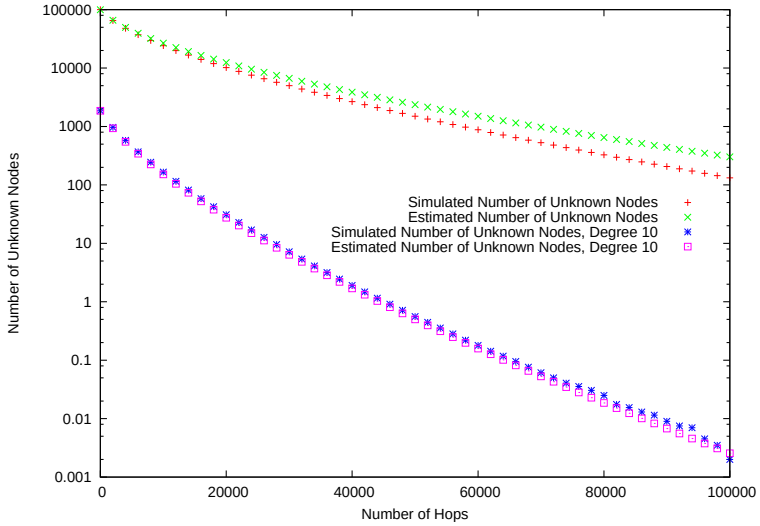


Figure 5.12: Known nodes experiments and estimations results, log scale, Barabasi-Albert network.

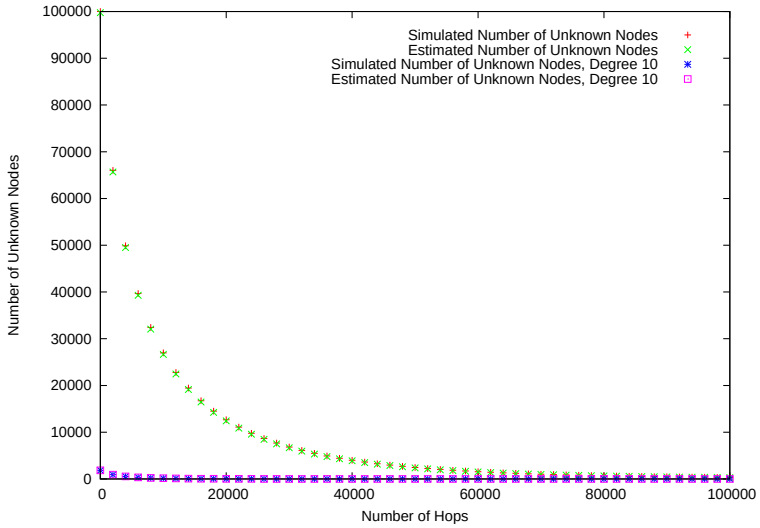


Figure 5.13: Known nodes experiments and estimations results, Newman network.

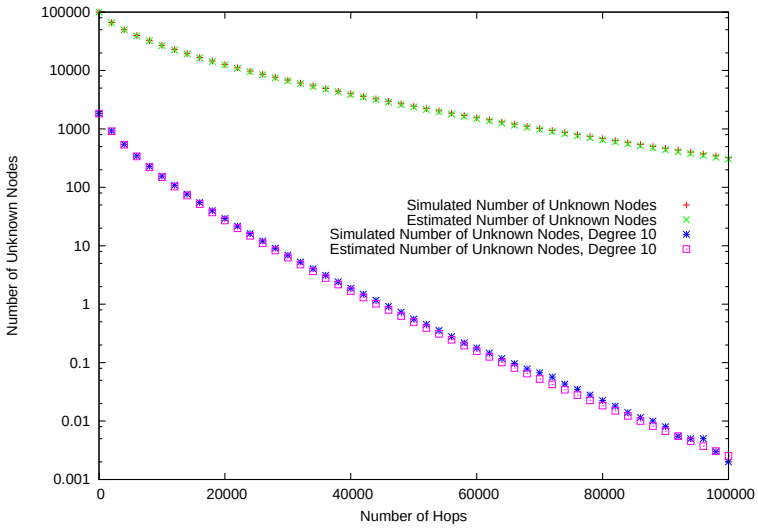


Figure 5.14: Known nodes experiments and estimations results, log scale, Newman network.

divergence in the Barabasi networks. For the first time, the model does not fit totally the experimental values. As we expected, the model gives similar values for Newman and Barabasi networks. Yet, the results obtained in our simulations for both networks differ. This leads to an important conclusion: *the behavior of random walks is not dictated only by the distribution degree, but also by how the network is built*. Hence, the model can not be totally accurate for all networks. Yet, even for the Barabasi network the error is proportionally small, compared with the network size.

5.2.6 Average Search Length

Using the previous metrics we are now able to find an estimation of the average search length, $\overline{l}$. We assume that each search starts from some random node, and tries to locate another random node by traversing the network following a random walk.

First, we compute the *success probability* at each hop l, P^l_{Success}, as the probability of finding the node looked for at that hop. It can be obtained from

the number of known nodes at hop l, C^l, using the following expression:

$$P^l_{\text{Success}} = \frac{C^l - C^{l-1}}{N - C^{l-1}} \tag{5.14}$$

$N-C^l$ is the number of nodes that are still unknown at hop l, which are also the only possible candidates to be the destination node. $C^l - C^{l-1}$ represents the number of new nodes that will be known at hop l, which can be taken as the number of trials at that hop. Finally, we get the success probability as the number of trials divided by the number of candidates.

Then, $\overline{l}$ is given by the following expression

$$\overline{l} = \sum_{l}^{\infty} l P^l_{\text{Finish}}, \tag{5.15}$$

where P^l_{Finish} is the probability that the search is finished at hop l; that is, the probability that the search is successful at that hop and has failed during the previous $l-1$ hops. This can be computed as

$$P^l_{\text{Finish}} = P^l_{\text{Success}} \prod_{i=0}^{l-1} (1 - P^i_{\text{Success}}) \tag{5.16}$$

Using Eq. 5.14 we can rewrite P^l_{Finish} as

$$P^l_{\text{Finish}} = \frac{C^l - C^{l-1}}{N}, \tag{5.17}$$

and so $\overline{l}$ would be computed as

$$\overline{l} = \frac{1}{N} \sum_{l}^{\infty} l(C^l - C^{l-1}). \tag{5.18}$$

We have checked the estimations of the average search length $\overline{l}$. Our experiments were run on networks of different sizes: 10^4, $2.5 \cdot 10^4$, $5 \cdot 10^4$, 10^5, $2 \cdot 10^5$, $5 \cdot 10^5$ and 10^6 nodes. For each network size, three different networks were built (Erdos-Renyi, Barabasi and Newman). 10^4 searches were run on each network for each experiment. For each search two nodes, a source and a destination, were chosen uniformly at random among the members of the network (the

source and destination could even be the same node). Then, starting from the source node, a random walk traversed the network until the destination node was found. Finally, the number of hops performed was annotated. The experimental results for each network, along with the predictions of the model, are shown in Figs. 5.15, 5.17 and 5.19. The same results, in logarithmic scale are shown in Figs. 5.16, 5.18 and 5.20.

The results observed seem to prove that the model can give a good approximation to the real values. And, although a perfect accuracy is not achieved, the error rate is kept constant as the network size grows. The worst results are obtained for Barabasi networks, where the error is close to 8.5%.

But the experiments show also that a total accuracy can not be obtained in general using only the degree distribution. We can conclude that from the experiments with Barabasi and Newman networks. Recall that Newman networks are built using the same degree distribution that a Barabasi network. As the degree distribution is the only input to compute $\overline{l}$, Eq. 5.18 returns very similar estimations for networks of both types. Yet, the experiments show that the real average search lengths are different, depending on the network type (Barabasi networks require less hops in average).

Hence, we can conclude *that not only the degree distribution, but also the mechanisms used to build the network have a strong influence on the behavior of searches*. Thus, it is not possible to obtain a perfect estimation of the average search length using only the degree distribution.

5.3 Average Time to Solve Searches

Here we explain the second part of our model, a proposal based on Queuing Theory to compute the average time that takes to solve searches in a network, given the capacities and degrees of its peers. First, we will give a reminder of some basics of Queuing Theory [72]. Then, we introduce our model and discuss the results obtained by simulation.

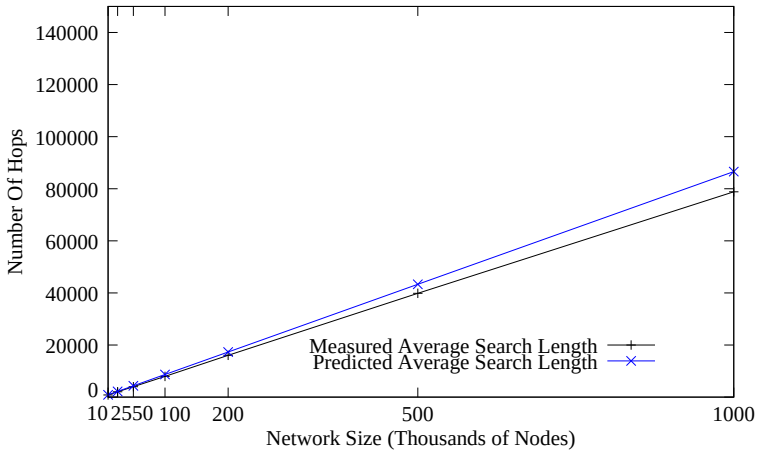


Figure 5.15: Average Search Length, Barabasi Networks.

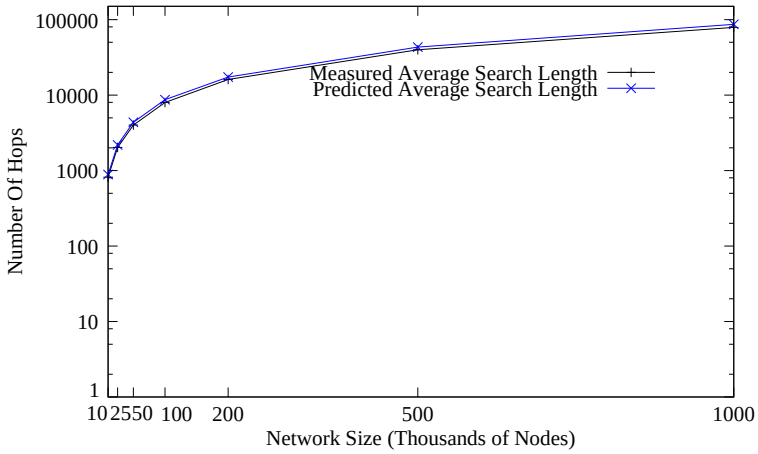


Figure 5.16: Average Search Length, Log Scale, Barabasi Networks.

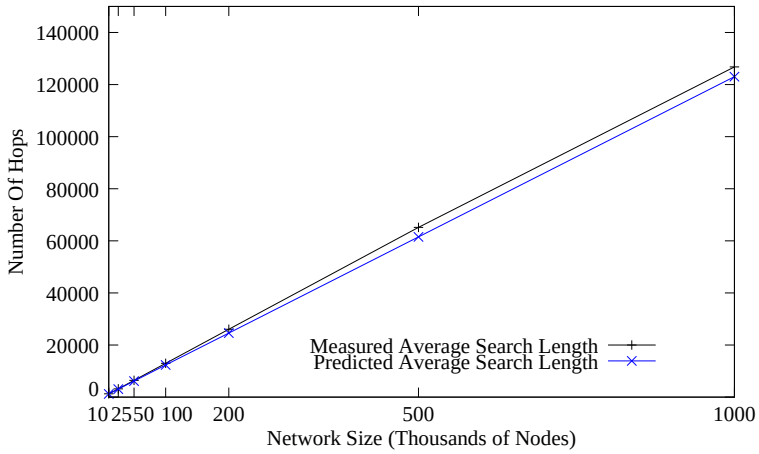


Figure 5.17: Average Search Length, Erdos-Renyi Networks.

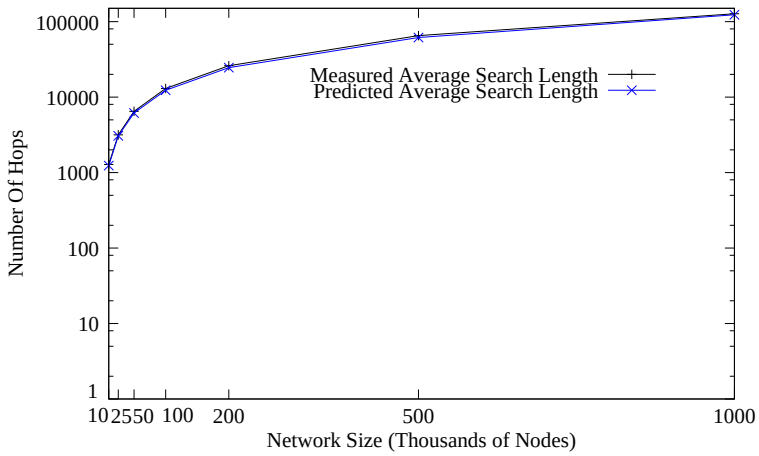


Figure 5.18: Average Search Length, Log Scale, Erdos-Renyi Networks.

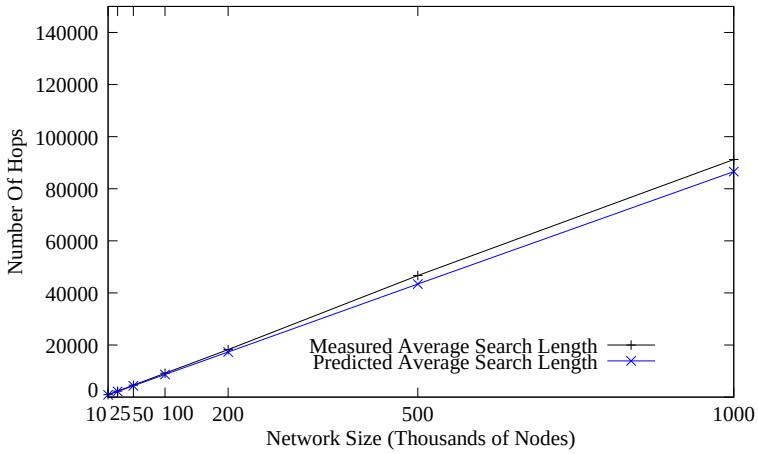


Figure 5.19: Average Search Length, Newman Networks.

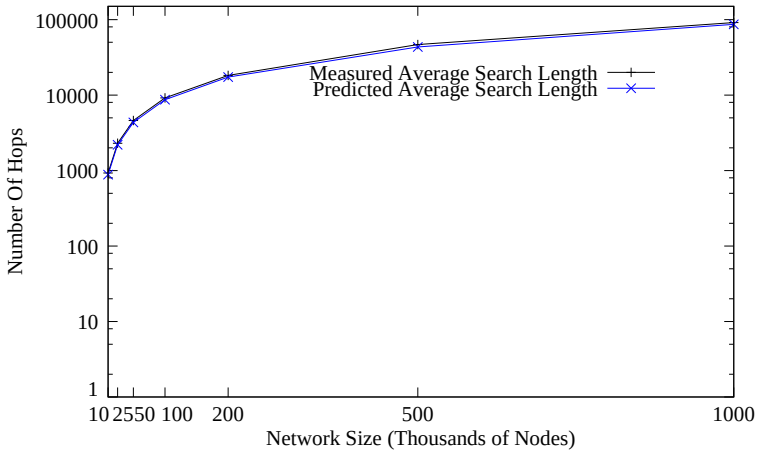


Figure 5.20: Average Search Length, Log Scale, Newman Networks.

5.3.1 Basic Queuing Theory: Little's Law, Utilization, Residence Time

This section is just a reminder of *Little's Law*. First, we need to define some parameters:

r = Number of items in the system (waiting and being served).
λ = Arrival rate, mean number of arrivals per second (load).
T_w = Average waiting time (time in queue).
T_s = Average service time.
T_r = Average residence time (time in system), $T_r = T_s + T_w$.
ρ = Utilization rate (time the system is busy).

Then, *Little's Law* says that:

$$r = \lambda \times T_r \tag{5.19}$$

Besides, there is a relationship (that is true in any single server system) that relates utilization, load and service time:

$$\rho = \lambda \times T_s \tag{5.20}$$

Finally, when the system is M/M/1, we have that:

$$T_r = \frac{T_s}{1-\rho} \tag{5.21}$$

As we will see in next sections, this expression is useful as nodes in our network are M/M/1 queues.

5.3.2 Basic network Queuing Theory: Jackson's Theorem

Here, we will introduce the *Jackson Theorem* [82], that can be used to analyze a network of queues. It makes the following assumptions:

- The queuing network consists of N nodes, each of which provides an independent exponential service.

- Items arriving from outside the system to any one of the nodes arrive with a Poisson rate.
- Once served at a node, an item goes to one of the other nodes with a fixed probability, or out of the system.

Jackson's Theorem states that in such a network of queues, *each node is an independent queuing system that can be analyzed using the M/M/1 model.*

5.3.3 Modeling the network

Fist, we define some basic concepts:

N = Size of the network.
r = Searches in the the system (waiting and being served).
T_r = Average residence time in the system.
γ = Total load injected to the system (generated by the nodes).
λ_S = Total load on all links (load due to search messages).
Λ = Total load on the system ($\Lambda = \lambda_S + \gamma$).
S = Total number of links.
λ_i = Load on link i ($\lambda_S = \sum_i^S \lambda_i$).
Λ_j = Load on node j ($\Lambda = \sum_j^N \Lambda_j$).
k_j = Degree of node j ($S = \sum_j^N k_j$).
r_j = Searches in node j ($r = \sum_j^N r_j$).
T_{rj} = Average residence time in node j.
T_{sj} = Average service time in node j.
ρ_j = Utilization rate of node j.

Our goal is to compute T_r, that is, the average time that takes to solve one search. We are assuming that the network fulfills all the assumptions of the *Jackson's Theorem* (see section 5.3.2). We must assume also that no node is overloaded (if not, the network would never get a stable state). That is, $\forall\, j\ \Lambda_j < 1/T_{sj}$.

Let $\overline{l}$ be the average number of hops for each search. Then, each search will be processed, on average, $1 + \overline{l}$ times (once at the origin node, and once at each visited node). Using this, we can express the total load on the system, Λ as:

$$\Lambda = (1 + \overline{l})\, \gamma \tag{5.22}$$

By definition, $\Lambda = \lambda_S + \gamma$. Combining this with Eq. 5.22 we can compute $\overline{l}$ as:

$$\overline{l} = \frac{\lambda_S + \gamma}{\gamma} - 1 = \frac{\lambda_S}{\gamma} \tag{5.23}$$

Assuming that the load on all links is approximately the same[1]: $\forall i \, \lambda_i = \tau$, then $\lambda_S = \sum_i^S \lambda_i = S\tau$ and so:

$$\overline{l} = \frac{S\tau}{\gamma} \tag{5.24}$$

Now we compute load on node j, Λ_j, as:

$$\Lambda_j = k_j\tau + \frac{\gamma}{N} = k_j\frac{\overline{l}\gamma}{S} + \frac{\gamma}{N} \tag{5.25}$$

The first addend of Eq.5.25 refers to the load due to search messages, and the second addend to the load that is started on the node itself (assuming that all nodes generate approximately the same load).

The next step is to get T_{rj}. By *Jackson's Theorem*, we know that each node behaves as a M/M/1 queue. Thus, we can apply Eq. 5.21 and so obtain that:

$$T_{rj} = \frac{T_{sj}}{1 - \rho_j} \tag{5.26}$$

that using Eq. 5.20 becomes:

$$T_{rj} = \frac{T_{sj}}{1 - \Lambda_j T_{sj}} \tag{5.27}$$

To compute T_r, we apply *Little's Law* (see section 5.3.1), obtaining:

$$r = \gamma \times T_r \tag{5.28}$$

It is straightforward that the total number of searches resident in the network r is the sum of the searches in all nodes r_j. Hence, we have that:

$$r = \sum_j^N r_j \tag{5.29}$$

[1]Our experiments show a linear relation between degree and load.

To obtain r_j, we apply *Little's Law* again, this time to each node j individually:

$$r_j \;=\; \Lambda_j \;\times\; T_{rj} \tag{5.30}$$

Applying to Eq. 5.28 the Eqs.5.29, 5.30 and 5.27 we have:

$$T_r \;=\; \frac{r}{\gamma} \;=\; \frac{\sum_j^N r_j}{\gamma} \;=\; \frac{1}{\gamma}\sum_j^N \Lambda_j T_{rj} \;=\; \frac{1}{\gamma}\sum_j^N \frac{\Lambda_j T_{sj}}{1-\Lambda_j T_{sj}} \tag{5.31}$$

Finally we can develop the previous expression using Eq. 5.25, so we finally obtain an equation that computes **the average searches time** using the topology (given by the k_j), the nodes capacities (contained in T_{sj}) and the load injected to the system (γ):

$$T_r \;=\; \frac{1}{\gamma}\sum_j^N \frac{T_{sj}\left(k_j\frac{\bar{l}\gamma}{S}+\frac{\gamma}{N}\right)}{1-T_{sj}\left(k_j\frac{\bar{l}\gamma}{S}+\frac{\gamma}{N}\right)} \tag{5.32}$$

This expression can be reduced to:

$$T_r \;=\; \sum_j^N \left(\frac{S}{T_{sj}\left(k_j\bar{l}+S/N\right)}-\gamma\right)^{-1} \tag{5.33}$$

5.3.4 Simulation Results

We have evaluated the goodness of the previous expressions by simulations. Each simulation tests a network of nodes, where each node has a certain capacity to process searches. Processing means to check if the resource looked for is known by the node itself or some of its neighbors.

A search can have been started locally by the node, or have reached it by means of a search message. When a node starts a new search, it first checks if it knows the resource asked for, if so the search has finished successfully. If not, a search message is created, containing the following info: the resource, the search origin node and the TTL. Then, the message is forwarded to some neighbor chosen at random. When a search message is processed, if the resource

Table 5.1: Capacities distribution for model simulations

Capacity Level	Percentage of nodes	Processing capacity C_j
1x	20 %	1
10x	45 %	10
100x	30 %	100
1000x	4.9 %	1000
10000x	0.1 %	10000

is known, the search is accounted as successfully finished. If not, the TTL of the message is decreased. If the TTL reaches 0, the search is accounted as unsuccessful, in other case the message is forwarded to a randomly chosen neighbor.

Whenever some node is processing a search it is said to be *busy*. If some other search is started locally or arrives by means of a search message when the node is *busy*, then the search is enqueued (queues are FIFO). When the node has finished processing a search, it checks its queue. If it is not empty, then it extracts the next search in the queue and process it.

For these experiments, all nodes have the same number of resources w ($w = 10000$). Each resource is hold by one and only one node. All resources are equally likely to be looked for (all are equally popular).

The network size is 10000 nodes. The capacity distribution chosen is the one depicted in Table 5.1. This distribution is derived from the measured bandwidth distributions of Gnutella nodes reported in [68], so we deem is a realistic setup of the capacities of peers in real P2P networks.

The service times to process searches in node j, T_{sj}, follow a exponential distribution, with average λ_j ($T_{sj} \sim \text{Exponential}(\lambda_j)$), where λ_j is computed as the number of resources known (those hold by the peer itself and its neighbors,

$(k_j + 1)w$) divided by the capacity C_j:

$$\lambda_j \;=\; \frac{(k_j+1)w}{C_j} \tag{5.34}$$

The load for all the network γ, of each experiment is computed taking into account that no node must be overloaded ($\forall\, j\ \Lambda_j \;<\; 1/T_{sj}$). Remember that Eq. 5.25 defines the relation between γ and the load on node j, Λ_j. So we have:

$$\Lambda_j \;=\; k_j\frac{\overline{l}\gamma}{S} + \frac{\gamma}{N} \;<\; \frac{1}{T_{sj}} \tag{5.35}$$

And so we can conclude that, for all nodes j the following must hold:

$$\gamma \;<\; \frac{SN}{T_{sj}(k_j\overline{l}N + S)} \tag{5.36}$$

We denote $\gamma_{\mathrm{Ov}j}$ as the load that would overload node j:

$$\gamma_{\mathrm{Ov}j} \;=\; \frac{SN}{T_{sj}(k_j\overline{l}N + S)} \tag{5.37}$$

Using the previous values we define γ_{Limit}, the minimum load that would overload the network:

$$\gamma_{\mathrm{Limit}} \;=\; \min_j\{\gamma_{\mathrm{Ov}j}\} \tag{5.38}$$

For each network, six different global loads were tested: $0.15 \times \gamma_{\mathrm{Limit}}$, $0.3 \times \gamma_{\mathrm{Limit}}$, $0.45 \times \gamma_{\mathrm{Limit}}$, $0.6 \times \gamma_{\mathrm{Limit}}$, $0.75 \times \gamma_{\mathrm{Limit}}$ and $0.9 \times \gamma_{\mathrm{Limit}}$.

As before, three different networks are tested: an Erdos-Renyi, a Barabasi and a Newman network. For each network and load a experiment is run, and three values are obtained:

- *Measured average searches time T_r.* It is the time that searches took, in average, to be solved, obtained from the experiment.

- *T_r estimated using predicted load on nodes.* Estimation of the T_r using Eq. 5.31, where the load on each node j, Λ_j, is predicted using its degree k_j and the estimated average search length $\overline{l}$ obtained from applying the model explained in the previous section.

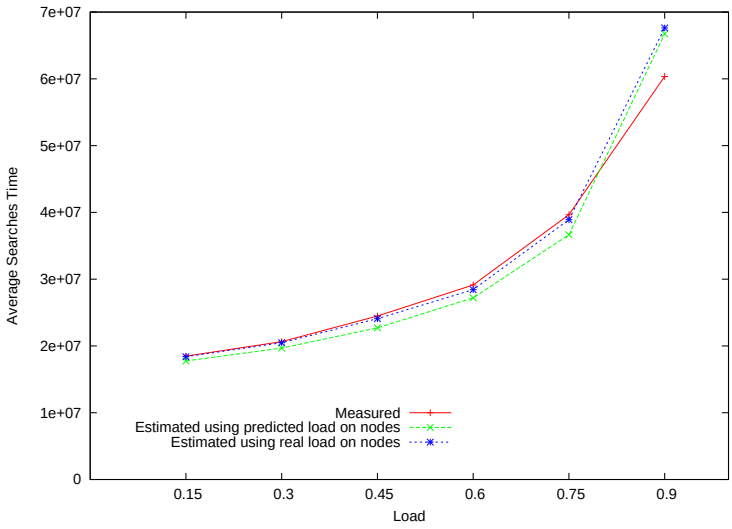


Figure 5.21: Average searches time, Erdos-Renyi network.

- T_r *estimated using real load on nodes*. Estimation of the T_r using Eq. 5.31, where the load on each node j, Λ_j, is obtained straight from the experiments.

Results are shown in Figs. 5.21, 5.22 and 5.23.

As we can see, predictions for Erdos-Renyi and Newman networks are quite accurate. There are some differences between the T_r measured and estimated using the predicted $\overline{l}$, but we deem this is due to the inaccuracy of the $\overline{l}$ prediction itself. The same holds for the values predicted for Barabasi networks and the results actually obtained, as they T_r estimated using the actual load on nodes is very close to the experimental results (remember that the error of the estimated predicted $\overline{l}$ for Barabasi networks was the highest).

5.3.5 Model for Centralized Topology

The previous model should be adapted for the special case when the topology is centralized. In this section we present our proposal for this special situation. Let N be the size of the network, let z be the number of central nodes and $N-z$ the number of peripheral peers.

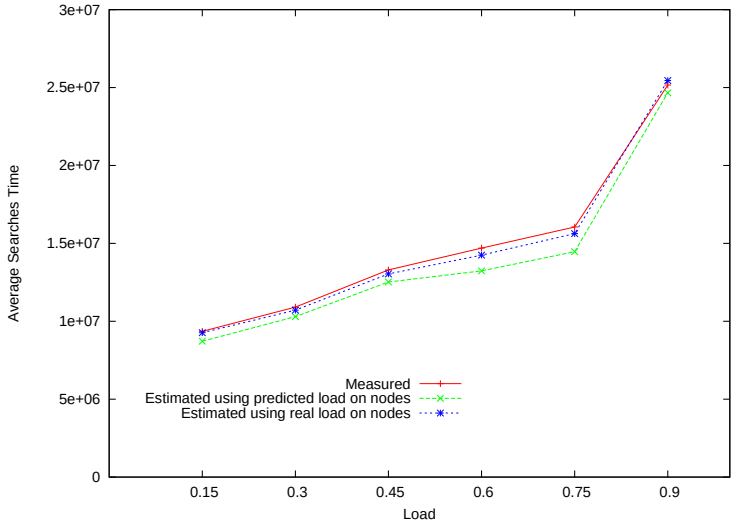


Figure 5.22: Average searches time, Newman network.

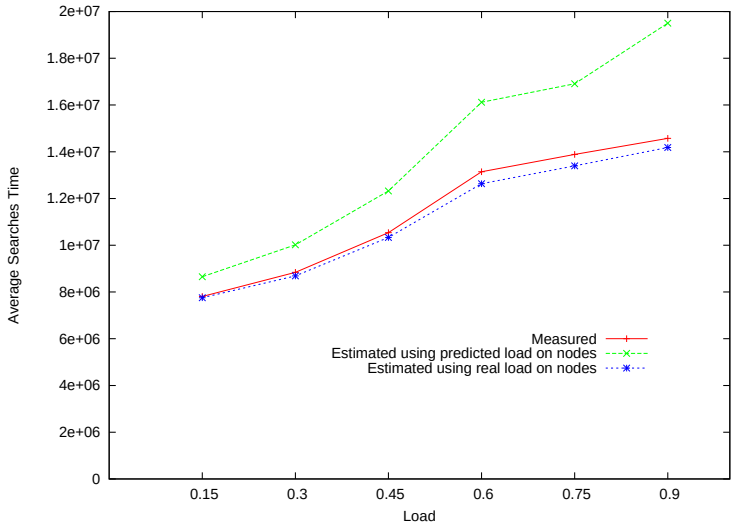


Figure 5.23: Average searches time, Barabasi network.

In a starlike topology, central nodes know all the resources in the system, hence we have that:

- Searches started on central nodes can be solved locally as central nodes known all resources.
- Central nodes can reply to all incoming search messages. Thus, they do not forward any search message.

In conclusion, all network load due to search messages is generated by the *peripheral* peers, and only central nodes receive search messages. Besides, all searches are solved in 1 hop, $\overline{l} = 1$[2].

Let j be a peripheral node. Then, by Eq. 5.25 Λ_j, the load on that node, is defined by:

$$\Lambda_j = k_j \frac{\overline{l}\gamma}{S} + \frac{\gamma}{N} \tag{5.39}$$

The first addend refers to the load received through j connections, and the second to the load generated locally.

The node j has a connection with all the other nodes in the network. So $k_j = N - 1$. But as we stated before, only peripheral nodes create search traffic, so the node will only receive searches from the $N - z$ non-central peers. Thus, we can replace k_j by $N - z$ in the Λ_j definition.

Remember that γ represents the global load generated on the system (all locally started searches), and γ/N the load generated by each node. Remember also that $\gamma\overline{l}$ is the load on the network due to search messages. As $\overline{l} = 1$, the load on the network becomes γ. But in fact central nodes *do not* generate search messages, so the load on the network becomes then $\frac{\gamma}{N}(N - z)$.

Finally, S is the sum of degrees of all nodes in the network ($S = \sum_j^N k_j$). By dividing the network load by S, we obtain the load that one node receives through each one of its connections. But in a centralized topology only the z

[2]More precisely, $l \leq 1$, as some queries started at no peripheral nodes can be solved locally, without creating a new search message. But normally $N \gg z$ and so $l \approx 1$

central nodes receive search messages, and only through their $N - z$ connections to peripheral nodes. Thus, S should be replaced by $z(N - z)$.

Making all these replacements on the definition of Λ_j, we obtain that:

$$\Lambda_j = \frac{\gamma}{z} \tag{5.40}$$

5.3.6 Relationship Between Performance, Degree and Capacity

The model developed in section 5.3.3 can be used to study the properties and behavior of a network under different circumstances.

Here, using the expressions described above, we will try to study how a proper distribution of capacities and degree can lead to better performances. More precisely, we will prove that *for any degree distribution, the minimum T_r is reached when the nodes with the greatest degrees are those with the greatest capacities also.* That is, if $\forall i, j; k_j > k_i \Rightarrow C_j > C_i$, the T_r of the network will be lesser than the T_r obtained with any other distribution of capacities that does not accomplishes the previous condition.

To prove this, first we need to set how the average service time T_{si} of each node i is computed. In our model, the T_{si} is proportional to the degree k_i: the greater the degree, the more resources are known and the more time it takes to check them all at each search. Besides, that same time T_{si} is inversely proportional to i's capacity C_i. Thus, we define T_{si} as:

$$T_{si} = \frac{Rk_i}{C_i} \tag{5.41}$$

Let G be a network with N nodes. Let i and j two nodes of the network with degrees k_i and k_j respectively, where $k_i < k_j$. Let C_1 and C_2 two capacities such as $C_1 > C_2$. Finally, let T_r the average searches time when C_1 is assigned to i and C_2 assigned to j (the node with a lesser degree has a greater capacity), and T'_r when that assignation is reversed. Then, our goal is to prove that $T'_r < T_r$.

First, using Eq. 5.41, we define the following service times:

$$T_{si} = \frac{Rk_i}{C_1} \; ; \; T'_{si} = \frac{Rk_i}{C_2} \; ; \; T_{sj} = \frac{Rk_j}{C_2} \; ; \; T'_{sj} = \frac{Rk_j}{C_1} \tag{5.42}$$

That is, T_{si} is the service time of node i when it has the capacity C_1, and T'_{si} is the same time when the capacity is C_2. From these equations we extract some useful expressions:

$$T_{si}T_{sj} \;=\; \frac{R^2 k_i k_j}{C_1 C_2} \;=\; T'_{si}T'_{sj} \tag{5.43}$$

It is straightforward that $T_{si} < T'_{si}$ and $T'_{sj} < T_{sj}$. Besides, taking into account that $k_i < k_j$, we have that:

$$T'_{si} - T_{si} \;=\; k_i \frac{R^2(C_1 - C_2)}{C_1 C_2} \;<\; k_j \frac{R^2(C_1 - C_2)}{C_1 C_2} \;=\; T_{sj} - T'_{sj} \tag{5.44}$$

Now we take a look on Λ_i and Λ_j, the loads on both nodes. From Eq. 5.25 we can conclude that those loads do not change as their degrees are not changed either, nor the global load γ. Besides, from the same equation we can state that $\Lambda_i < \Lambda_j$. Using this and the Eq. 5.44, we say that:

$$\Lambda_i T'_{si} - \Lambda_i T_{si} \;<\; \Lambda_j T_{sj} - \Lambda_j T'_{sj} \tag{5.45}$$

and so:

$$\Lambda_i T'_{si} + \Lambda_j T'_{sj} \;<\; \Lambda_i T_{si} + \Lambda_j T_{sj} \tag{5.46}$$

To compute the T_r and T'_r values, we use Eq. 5.31. Thus, we obtain that:

$$T_r \;=\; \frac{1}{\gamma}\left(\sum_{h\neq i, h\neq j}^{N} r_h + r_i + r_j\right) \;;\; T'_r \;=\; \frac{1}{\gamma}\left(\sum_{h\neq i, h\neq j}^{N} r_h + r'_i + r'_j\right) \tag{5.47}$$

The r_h keeps constant for any node h that is not i nor j (its degree, load and capacity are just the same). Hence, $T'_r < T_r$ if and only if $r_i + r_j > r'_i + r'_j$. From Eqs. 5.27 and 5.30 we know that:

$$r_i + r_j \;=\; \frac{\Lambda_i T_{si}}{1 - \Lambda_i T_{si}} + \frac{\Lambda_j T_{sj}}{1 - \Lambda_j T_{sj}} \tag{5.48}$$

$$r'_i + r'_j \;=\; \frac{\Lambda_i T'_{si}}{1 - \Lambda_i T'_{si}} + \frac{\Lambda_j T'_{sj}}{1 - \Lambda_j T'_{sj}} \tag{5.49}$$

Developing the previous expressions, and using Eq. 5.43, we have that:

$$r_i + r_j \;=\; \frac{-2\Lambda_i\Lambda_j T_{si}T_{sj} + \Lambda_i T_{si} + \Lambda_j T_{sj}}{1 + \Lambda_i\Lambda_j T_{si}T_{sj} - (\Lambda_i T_{si} + \Lambda_j T_{sj})} \tag{5.50}$$

$$r'_i + r'_j \;=\; \frac{-2\Lambda_i\Lambda_j T_{si}T_{sj} + \Lambda_i T'_{si} + \Lambda_j T'_{sj}}{1 + \Lambda_i\Lambda_j T_{si}T_{sj} - (\Lambda_i T'_{si} + \Lambda_j T'_{sj})} \tag{5.51}$$

Applying now Eq. 5.46 we can finally conclude that

$$r'_i + r'_j \;<\; r_i + r_j \tag{5.52}$$

and hence:

$$T'_r \;<\; T_r \tag{5.53}$$

Chapter 6

An Implementation of Cholvi's Proposal

Before designing DANTE, we created a real implementation of a P2P system whose nodes run the reconnection mechanism proposed by Cholvi *et. al-* [16] described in section 4.2. The results obtained influenced the design of DANTE, so there are many similarities between DANTE and this previous implementation.

Although the simulation results shown in [16, 17] seemed to confirm the validity of Cholvi's design, we were concerned about some potential problems we detected, already described in section 4.2. Basically, our main concern was how the reconnection mechanism could be implemented on a real P2P network, where not global knowledge is available: nodes do not know all the other peers present in the system, neither their state. Besides, in their model and simulations, Cholvi *et al.* assumed that searches follow shortest paths, as is done by Guimerà, and not random walks. We intended to check if in an scenario with random walks the conclusions of Guimerà about the relationship between topology, load and searches efficiency would still be valid.

To test the goodness of Cholvi's proposal we have implemented in Java a P2P system where nodes run a reconnection mechanism based on that proposal. Our results verified that the solution outlined in [16, 17] can be successfully translated to real P2P systems, where searches are run by walkers that follow random walks, and the information about other peers, required to perform new

reconnections, is also obtained using random walks.

As this is a P2P system, all nodes are identical, in the sense that they are running instances of the same software.

Nodes handle a number of *native* connections. The reconnection mechanism determines to which other peers the native connections must point to, depending on the attractiveness of those peers. Also, each node can have *foreign* connections, that are those established by other peers with that node. Thus, when one connection is *native* on one side of the link, is *foreign* on the other side. Nodes only change their *native* connections, not their *foreign* ones.

Each time a new reconnection is triggered, information about a set of peers in the system is collected by means of a random walk, bounded by a TTL. When the random walk finishes, the information gathered is sent back to the origin node. There, a value Π_j is computed for each peer j found in the random walk, using the definitions explained in section 4.2.

6.1 Nodes Components

The architecture of nodes is shown in Figure 6.1. In this section we introduce briefly the components in that architecture.

The **Communications Layer** takes care of the sending and reception of messages. Messages are transmitted by UDP. A simple ACK mechanism is implemented by this module for a better (not total) reliability.

The **Packet Listener** module is constantly listening for new incoming messages (given by the **Communications Layer** module). Each message received is given to the **Messages Queue Set**, that stores it into one of the three FIFO queues that it contains, each one with a different priority. The **Connections Queue** stores messages concerning the creation or closing of connections. It has the greatest priority, so messages in this queue will be processed before any other messages in the other queues. The **Searches Results Queue** stores the messages with the results of the resource searches. It has intermediate priority. Finally, the **Searches Petitions Queue** holds the messages carrying the

lookup for resources. It has the minimum priority.

The **Message Attender Thread Pool** component contains a set of threads that pick the messages from the queues and execute the tasks associated to them, excepting the messages related with connections. Those messages are processed by the **Connected Nodes Set** module, that is in charge of the functionality related with the opening and closing of connections.

Reconnections are triggered by the **Reconnecter**. The **Node Finder** is then called to obtain a list of new neighbors candidates from the system. When that list of candidates is ready, the **Dynamic Network Topology Kernel** chooses the peers that the node should try to connect to, changing its *native* connections. The **Connections Accepter**, on the other hand, is the module that decides if the connection petitions from other peers must be accepted or not, implementing a certain politic (the default policy is to accept all connections).

When a search message (for some resource or for peers) is received by the node, the **Forward Resolver** is in charge of deciding to which neighbor the search must be forwarded to.

Finally, the **Resource Finder** takes care of resources lookups. It creates and launches the search messages, and waits for the results.

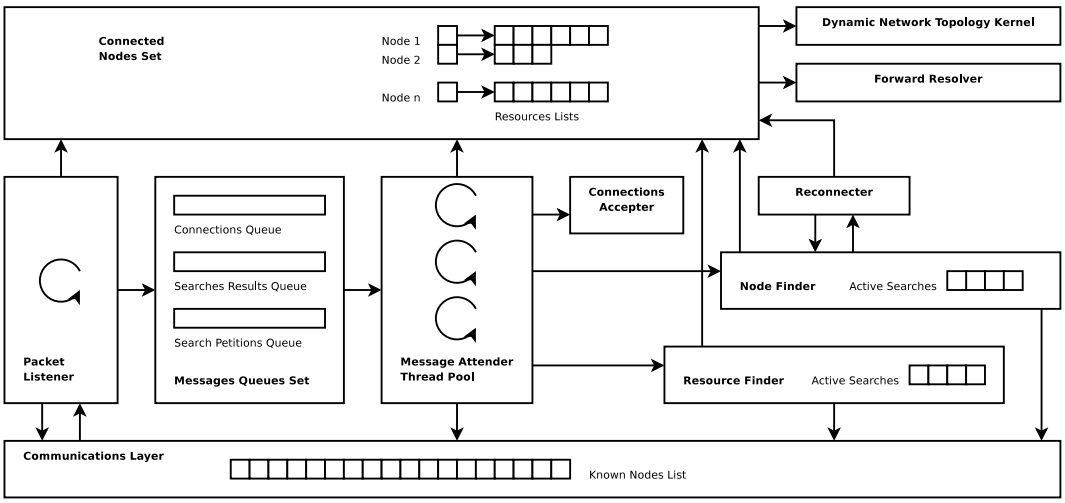


Figure 6.1: Nodes Architecture.

6.2 Protocol Messages

Communications are done through messages carried by UDP packets, complemented by an ACK mechanism to provide a certain reliability. There are three types of messages:

- *Connection Messages.* Used to ask for the creation of an overlay connection between nodes, to accept or decline the petition, and to close connections: *CONNECT, ACCEPT, REJECT, DISCONNECT.*

- *Nodes Search Messages.* Used to perform lookups for candidate peers for reconnections, and to send back the list of found peers when the search is over: *LOOK_FOR_NODES, NODES_FOUND.*

- *Resources Search Messages.* Used for resources searching, and to communicate whether the search was successful or not: *LOOK_FOR_RESOURCE, RESOURCE_FOUND, RESOURCE_NOT_FOUND.*

6.3 Experimental Results

A set of experiments were run to study the behavior of the system, to check if the results were as predicted by Guimerà.

6.3.1 Configuration

Experiments were run with 42 nodes, in a small cluster of 7 PCs. A *Subscription Service* was run in another PC. Nodes used this service at start time to register themselves as alive nodes, and to know other nodes in the system.

Each node hosts 2000 resources. Each resource is hold by one and only one node. As stated before, all resources are equally popular.

Searches are started by a *virtual user*, one per node. Periodically, this user starts a new search for a resource chosen at random. The searches frequency sets the overall load on the network.

In each experiment, a *capacity threshold* is set for all nodes (the same for all of them). If some node receives a number of searches per minute greater than its threshold, then the node is said to be *congested*. The threshold parameter should, of course, represent the real capacity of the node. However, it can be also used to force the network to form certain topologies whatever the network load. The goal is to be able to compare different topologies under different loads.

For example, in some experiments we set the threshold to 0. As a result, all nodes are assumed to be congested whatever the load. This forces the network to form a random overlay topology. On the other hand, in other experiments we set the threshold to a value high enough so no node is ever *considered to be* congested, no matter how many searches it processes. Thus, the network will form a centralized topology regardless of the load.

Reconnections are triggered every 30 seconds. Nodes handle 3 *native* connections each.

Each experiment lasts 2 hours. Results concerning the 15 first minutes and 45 last minutes of experiments are ignored.

6.3.2 Topology Adaptation

Our first goal is to check that indeed the network adapts to the load: it must form a random topology for low loads, a clustered topology (with hubs) for middle loads and a centralized topology for high loads. We have run three experiments with different loads: 5, 10 and 15 loads per minute and node. The congestion threshold was set to 10 searches for all experiments.

The initial topology is random in the three experiments (nodes choose their first neighbors randomly). The resulting topologies are shown in Figs. 6.2, 6.3 and 6.4.

Figure 6.2 shows the topology under a low load of 5 queries started by each node per minute. The resulting topology is centralized, with three central nodes connected to (almost) all other peers in the network. In the following figure, fig 6.3, we see how the topology has evolved to a clustered form, where some

nodes have many connections without being central. For that experiment, the load was set to 10 queries per minute and node. Finally, the topology resulting under a high load of 15 queries per minute and node is displayed in Fig. 6.4. As expected, the topology obtained has a random form, where all nodes have a quite similar number of neighbors.

Thus, we can conclude that the network topology behaves as intended.

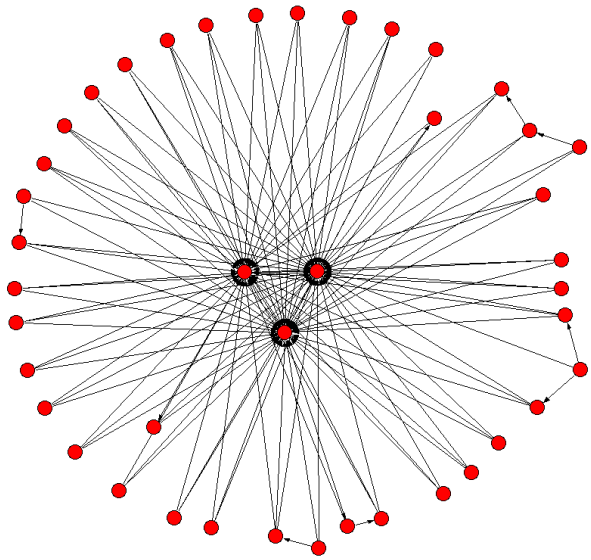


Figure 6.2: Topology under load of 5 queries per minute and node.

6.3.3 Searches Performance

To evaluate the performance of searches, a set of experiments were run with different loads and thresholds. In Figs. 6.5 and 6.6 we can see the results of those experiments. Note that some experiments were run with threshold 0. As shown above, that forces the network to keep a random topology. On the other hand, the threshold 1000000 is big enough to make the network to evolve to a centralized topology whatever the load used. In experiments with intermediate thresholds, the topology is centralized with low load, and moves to a random

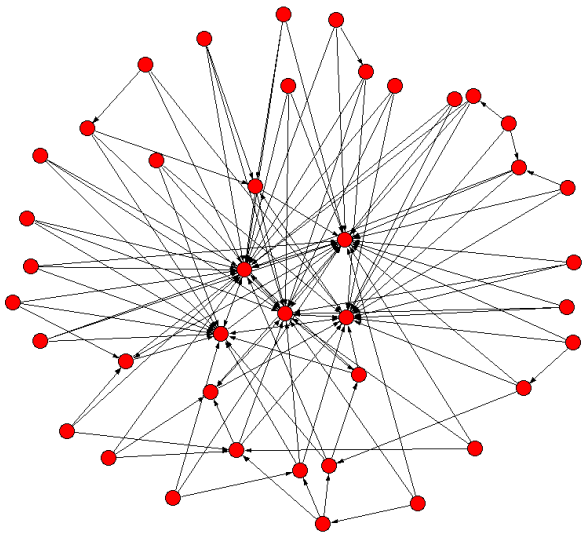


Figure 6.3: Topology under load of 10 queries per minute and node.

state as the load is increased. It is important to note that those thresholds are not related with the *real* capacity of nodes. In real systems it would be appropriate that each threshold is set in accordance with the capacity of the corresponding node (for our system, it seems that the more accurate threshold is 10). Yet for these experiments we intend to check the behavior of different topologies. Thus, we use the thresholds to force the topologies we want to test.

First we analyze the average search times obtained, that is represented in Fig. 6.5. Each line on the graph represents the experiments with a certain threshold. We see that, for low loads, the experiments with threshold 0 (random networks) obtain the worst results, while the rest of experiments, that form centralized networks, obtain better times. This is expected, as in centralized systems searches are solved in just one hop.

However, for high loads the results are reversed. The experiments with threshold 0 have the best performance. The rest (save those for threshold 10) get worst times, greater as the topology is more centralized (bigger thresholds). This is due to the congestion on the central nodes or hubs. In general, we can

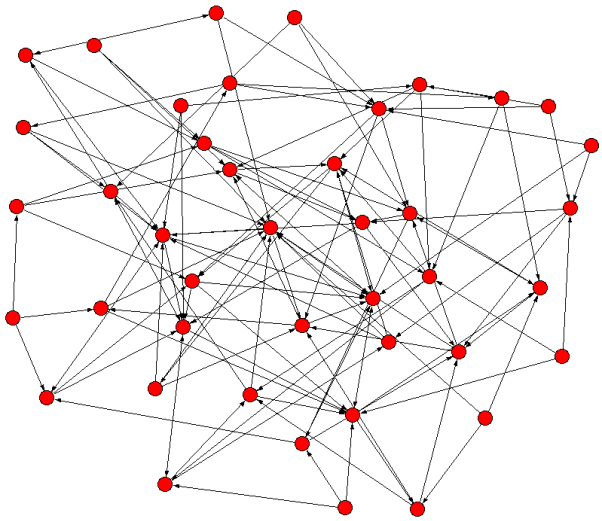


Figure 6.4: Topology under load of 15 queries per minute and node.

conclude that behavior of the network confirms Guimerà predictions.

It is worth to note the results for threshold 10. As it is explained above, this threshold is the one that best approaches to the real performance of nodes. We can see that its performance is always close to the optimal times for almost all the loads, so we can conclude that the algorithm indeed makes the network to form efficient topologies. The exception are the results obtained with load 6, where the threshold 10 has the worst performance. The reason, we deem, is that for that load well connected nodes receive many searches that they must enqueue. When one of those nodes gets overloaded, then its neighbors will disconnect from it and so the peer will lose much information about the resources in the system. Because of this, it will not be able to answer to many of the queries it has in queue, and so those queries will have to be forwarded to other peers where, with high probability, the same process will be repeated.

In Figure 6.6 we represent the average number of hops per search against the network load. Results are grouped, as before, by thresholds. There we can confirm that for very high thresholds the network keeps centralized, no matter

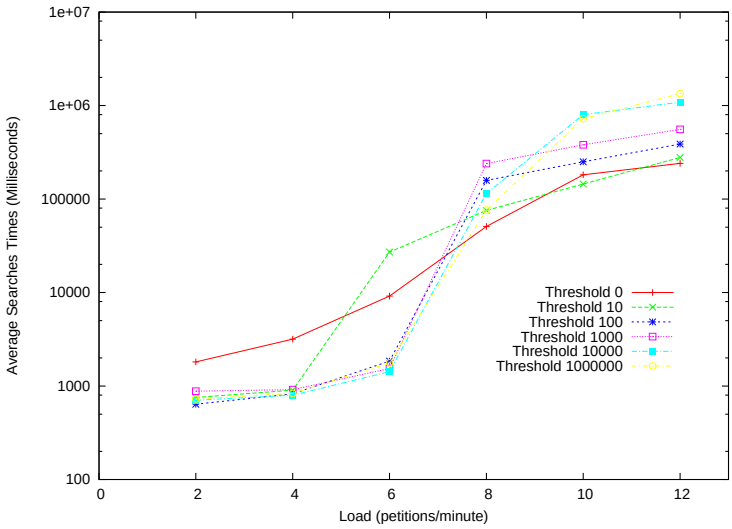


Figure 6.5: Average Searches Times.

the load. On the other hand, the other thresholds move suddenly to a random topology when the load is increased. That is, it seems that no intermediate topologies are obtained, or they appear only for certain, very small load ranges. This agrees with Guimerà work, that states that optimal topologies move from a central form to a random one very sharply.

6.4 Conclusions

From the results obtained with this implementation we extracted the following conclusions:

- Using a reconnection mechanism like the one explained in section 4.2 is feasible in real P2P systems where searches are performed by random walks.

- Information about other peers state, needed for the reconnections, can be gathered using random walks. Global knowledge is not mandatory.

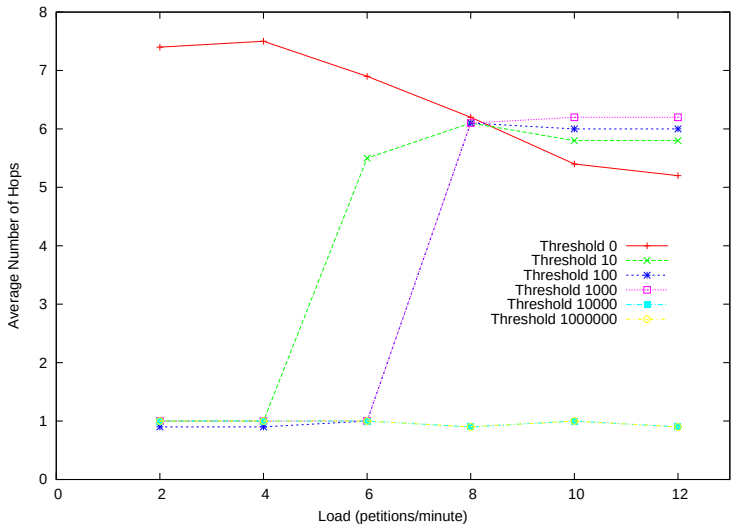


Figure 6.6: Average Number of Hops.

- Guimerà statements about the relation between topology, load and performance still hold in this scenario.

But it was also to detect some limitations on Cholvi's proposal:

- It is difficult to state a node's capacity by using a searches *threshold*. Depending on the circumstances, it can be very inaccurate.

- Strong changes on the topology are to be expected under certain loads. This is due to a very undesirable property of the metric used to measure each node attractiveness (Π): its value grows quickly as new peers connect to the node (so Π gets increased again and the node becomes even more attractive to other peers), but falls to the minimum value sharply, as soon as that node gets overloaded. After that, neighbors of that peer will tend to disconnect from it quickly. This process, when happens with certain frequency, prevents the topology to reach a stable state.

All this, along with the model presented in chapter 5 aided us to achieve some conclusions that were key on DANTE's design. Basically, the necessity to

find a reconnection kernel that did not need to set a fixed threshold, took into account nodes' load and avoided sharp changes on the topology.

Chapter 7

DANTE: A Self-Adapting P2P System

DANTE is our proposal of an unstructured P2P system able to adapt its topology to the network load conditions. It is based on the previous work of Cholvi *et al.* [16, 17], as described in section 4.2. DANTE uses the same idea of an *attachment kernel* to compute the probability that one node connects to some other peer in the network. Yet, it changes the first proposal of Cholvi so:

- Nodes' heterogeneity and load is taken into account.
- Fixed thresholds are not used, as they can be hard to compute.
- Sharp changes on topology are avoided. The *attractiveness* of each node is computed in a way that its value evolves in a continuous manner. This prevents other peers to change their links (pointing them to the node or disconnecting from it) abruptly.

In this chapter, DANTE's basis are explained. Also, we comment the results of some simulations that were have run to evaluate DANTE's performance.

7.1 DANTE Characteristics

First, we describe the main DANTE characteristics. DANTE is an unstructured P2P network. Each node can connect with any other node in the system to form

the overlay topology.

Besides, DANTE uses *one-hop replication*: each node maintains a list of the resources held by its neighbors in the overlay topology.

7.1.1 Searches Routing

Searches in DANTE are run using pure random walks. When some node starts a search for a certain resource, it first checks locally if some of its neighbors have it. If so, the search is finished. If not, a *search message* is built and send to some random neighbor. That message contains the resource looked for and the origin of the search, along with a *TTL* counter to limit the search scope and an unique ID that identifies the search.

When a search message arrives to some peer, that peer looks if any of its neighbors stores the resource. If so, it builds a *successful search* message that contains the peer that holds the resource and the search ID, and sends it to the origin node. If not, it checks the message TTL. If it is greater than 0, then the TTL is decreased in one unit and the search message is forwarded to some neighbor chosen at random. If the TTL is 0, then an *unsuccessful search* message is built and sent to the origin node, containing the search ID.

It is important to realize that, although DANTE in its actual version uses pure random walks for search routing, many other routing mechanisms could be used. The main property of DANTE is its ability to adapt its topology to the load, thanks to its reconnection mechanism. This mechanism does not depend on, and does not affect either, to the way messages are routed. So, for example, the improvements proposed by the research community to the pure random walk routing mechanism (see Sec. 3.2) could be easily combined with DANTE.

7.2 DANTE Self-Adaptation Mechanism

In DANTE each peer knows its own resources as well as the resources held by its neighbors. Based on this, it is easy to understand that nodes are more

interested on being connected to peers with many neighbors. Therefore, DANTE encourages peers to establish connections with high degree nodes. DANTE, in fact, aims to form highly clustered (even centralized) topologies. However, this holds only as long as highly connected nodes can handle all the incoming traffic. In brief, we can say that DANTE aims to achieve these goals:

- Make the overlay topology as centralized as possible.
- Avoid congestion in all peers.
- High degree nodes must be the ones with higher capacities, see section 5.3.6-

DANTE uses an algorithm that, when the network traffic is low, drives the network to a star-like overlay topology. Thus, searches can be answered in only one hop, since the central nodes know all the resources in the system. In turn, when the number of searches increases, well-connected nodes will become congested and their neighbors will start to disconnect from them. Hence, this will drive the network to a more random-like topology that, although it makes search messages to traverse longer paths to find some resource, will balance the load and perform better than by using a highly congested central node.

More specifically, in DANTE each node can establish connections to other nodes. We say that a connection is native for the establishing node and foreign for the accepting node. Nodes can change their native connections, but not their foreign ones. Furthermore, each node periodically runs a reconnection mechanism with which native connections are changed. This mechanism firstly obtains a list of potential candidates C to which it can connect. Then, it assigns a probability pi to each candidate i, and chooses candidates at random using their respective probabilities. Finally, the peer reconnects its native connections to the chosen candidates.

The probability assigned to a candidate $i \in C$ is based on its "attractiveness", denoted as Π_i. How Π_i is computed is the key of DANTE's reconnection functionality. We denote it the **attachment kernel**, and is defined by the

following expressions (some other kernel candidates were also considered, see Sec. 7.6.1 for a comparison). The Π_i value is computed using:

$$\Pi_i \ = \ k_i^{\gamma_i} \tag{7.1}$$

where k_i is the degree (number of neighbors) of peer i, and γ_i is computed as

$$\gamma_i \ = 2 \ \times \ c_{i_{norm}} \ \times \ (1 \ - \ t_{i_{norm}}) \tag{7.2}$$

$c_{i_{norm}}$ is the normalized processing capacity of node i, where the normalization is performed as follows. Let c_i be the capacity of node i, and $c_{max} = \max_{i \in C}\{c_i\}$. Then,

$$c_{i_{norm}} = \frac{c_i}{c_{max}} \tag{7.3}$$

it follows that $0 < c_{i_{norm}} \leq 1$, $\forall\, i$, where a larger $c_{i_{norm}}$ means that node i is more attractive as it has more capacity to process searches.

$t_{i_{norm}}$ represents the average time spent by a search at node i (time in queue plus processing time), normalized. The normalization is computed as follows. Let t_i be the mean search processing time of node i, $t_{max} = \max_{i \in C}\{t_i\}$ and $t_{min} = \min_{i \in C}\{t_i\}$. Then,

$$t_{i_{norm}} = \frac{t_i \ - \ t_{min}}{t_{max} \ - \ t_{min}}. \tag{7.4}$$

It is straightforward that $0 \leq t_{i_{norm}} \leq 1 \, \forall\, i$, where a lesser $t_{i_{norm}}$ means that node i is more attractive as it takes less time for searches to be served.

Finally, once the Π_i values are computed for all candidates in C, each candidate $i \ \in \ C$ is assigned a probability p_i of being chosen that is computed as

$$p_i = \frac{\Pi_i}{\sum_{j \in C} \Pi_j}. \tag{7.5}$$

Thanks to this reconnection mechanism, the system behaves in an adaptative manner, changing its topology to suit the load conditions. This behavior emerges as an effect of the individual work of nodes.

Higher capacity nodes tend naturally to be more connected, in some cases becoming a kind of *super-peers* (if they have enough capacity), to which all other

nodes are connected. But is important to make a distinction with the *super-peers* of systems like eDonkey. In DANTE, any node can become a central peer if it has enough capacity, as all nodes implement the same mechanism and algorithms. Then, the system itself is not dependant on any special nodes. If they are not present, or leave the network, then the topology adapts automatically to the circumstances. This is not the case of the eDonkey network, where *super-peers* run different algorithms than the other nodes, and where the network can not work without them.

7.2.1 Candidates Sampling

The reconnection mechanism of DANTE depends on the set of candidates C to which the node can connect. There are several mechanisms that could be used to build this list of candidates. For example, a *gossiping* based service like [39] could spread information about nodes in the network. Another solution is to make nodes to keep a *cache* of other peers in the network.

DANTE implements a third solution. Whenever a node starts a new reconnection, it launches a special *Look For Node* message, that traverses the network following a random walk with a bounded TTL. When the TTL expires, another message is sent to the origin node with the list of traversed peers. This list becomes the set of candidates C. This technique has small incidence on the network load, and Newman's results [58] show that the set obtained is a good sample of the overall network. Besides, well-connected peers will be traversed with higher probability that peers with low degree, and those nodes are potentially better candidates as each participant aims to connect to high-degree peers.

7.3 Simulation Framework

To evaluate DANTE's performance we run a series of simulations. In this section we describe how the simulation software is built.

All the software for these simulations was developed using the Java programming language. Simulations are run on a simple framework for discrete event simulation we have developed. The reason not to use other simulations software like ns-2 [26] or J-Sim [88] was the complexity of those frameworks, quite beyond our requirements.

Over this framework concepts like link, message, triggers, etc were programmed. Finally, node's functionality was developed on top of the previous software.

Simulations use the microsecond as the minimum unit of time. The capacity of each node is set by two parameters: *bandwidth* and *processing capacity*. Nodes perform tasks, like the processing of an incoming message or an internally started process (e.g., the triggering of a new reconnection). When performing some task the node is said to be *busy*. Any other pending task in the node is enqueued until the present task is finished. In some tasks, it is required to send a message to other peer. That is also a time consuming task.

The processing time t_{proc} depends on the tasks being performed. Tasks other than searching for a resource in the lists of known resources are assumed to take one unit of time. Searches for resources take a time proportional to the number of resources checked m and the node's processing capacity c_i, $t_{proc} = \frac{m}{c_i}$. Some tasks need to send a message. The duration of sending a message , t_{send}, depends on the node's bandwidth b_i and the packet size s, $t_{send} = \frac{s}{b_i}$. Finally, the time the node is busy, t_{busy}, for one task is computed as $t_{busy} = \max\{t_{proc}, t_{send}\}$. This time is not $t_{proc} + t_{send}$, because we assume that the sending of messages and the processing of searches run in a pipeline.

7.4 Simulations Configuration

Simulations are configured as follows. Nodes capacities and bandwidths are assigned following the distribution depicted in Table 7.1. This distribution is derived from the measured bandwidth distributions of Gnutella nodes reported in [68].

Table 7.1: Capacities and upload bandwidths distribution for DANTE simulations

Capacity level	Percentage of nodes	Processing capacity c_i	Bandwidth b_i
1x	20 %	0.1	0.01
10x	45 %	1	0.1
100x	30 %	10	1
1000x	4.9 %	100	10
10000x	0.1 %	1000	100

Each node starts a new search for a random resource periodically. The *time between searches*, *tbs*, is a parameter of the simulation that allows to set the load on the system. Each node holds 100 resources. All resources have the same popularity (no resource is more likely to be looked for than other). The *replication rate r* is another simulation parameter, that states the rate of nodes that hold each resource (in percent).

Nodes manage 10 *native* connections each. Reconnections are triggered every 30 seconds of virtual time. Nodes change 5 *native* connections at each reconnection. We assume there is an external service that provides peers, at start-up time, with a list of some other nodes present in the system. When some peer is started it chooses its initial neighbors randomly from the list provided by that service. Hence, all experiments start with a random topology. Similarly, if a *native* connection points to some node that leaves the network (is attacked or deactivated), that connection is redirected to another peer chosen at random from a list again obtained from the external service.

The *Look For Nodes* messages have a TTL of 30. Resource search messages have a TTL of 1000. Both values were chosen empirically. The first one proved to be enough to get a good sampling of the network, the second one allowed to obtain a high success rate, both for DANTE and Gia simulations.

7.5 Simulations Results

Finally, the simulations results are explained here. We have run different series of simulations to evaluate different aspects of DANTE's behavior.

7.5.1 Topology Evolution

First, we study how in DANTE the system is able to adapt itself, changing its topology as the (virtual) time passes. The results of two simulations are shown, with two different replication rates: $r = 0.01$ and $r = 0.05$. Both simulations are run with 10000 nodes (so $r = 0.01$ implies that each resource is hold by only one node) and a time between searches $tbs = 1$ second. Simulations were run for 60 minutes of virtual time. All searches finished successfully in both simulations.

In Figure 7.1 we see how the mean number of hops changed as the virtual time passed. In the X axis we represent the virtual time, in minutes. In the Y axis we represent the average number of hops that took to solve searches started during the corresponding minute of virtual time. The number of hops decreases readily as the time passed, until it is stabilized to 1 after some minutes (tens of reconnections). This means that the network has reached a centralized topology, starting from a random one, and all searches are solved in just one hop.

On the other hand, we see in Figure 7.2 how the topology evolution makes the average search time to decrease. DANTE builds an efficient topology, where searches are completed in very little time (about 30 milliseconds).

7.5.2 Scalability

An important concern on P2P systems is scalability. The system should be able to manage thousands of nodes, keeping the searches performance within acceptable bounds. In this section we comment the results of some simulations that show how in DANTE the overall performance can even be increased when new nodes are added.

The simulations of this section are run with 2000, 4000, 6000, 8000 and

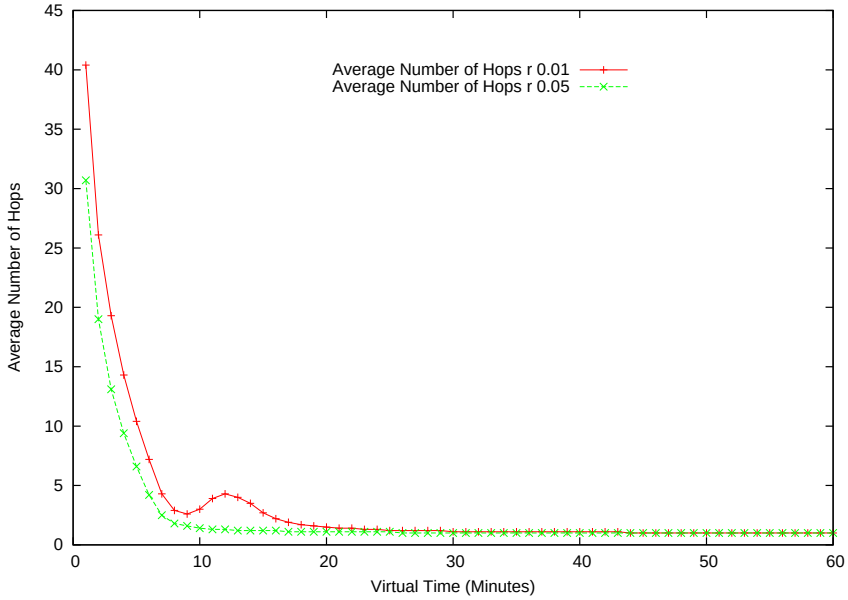


Figure 7.1: Number of hops as time passes.

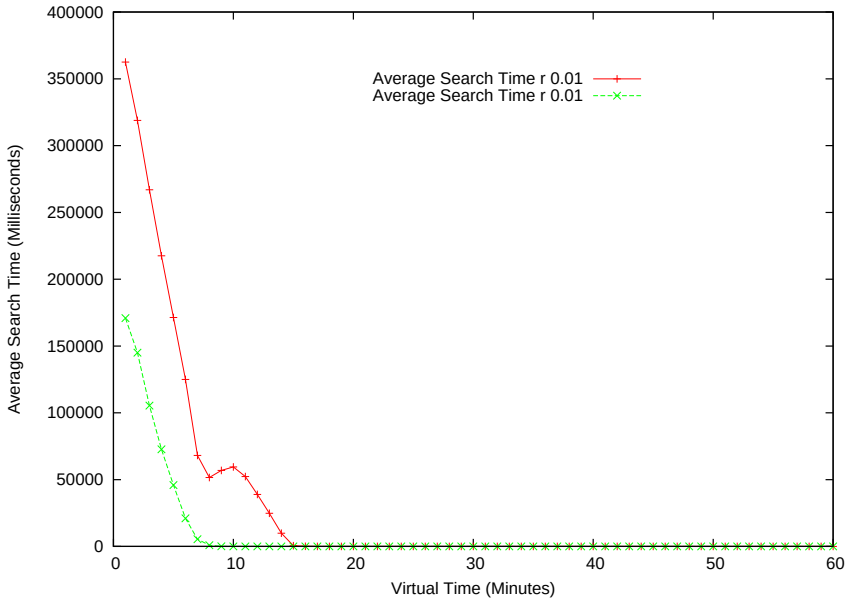


Figure 7.2: Search times as time passes.

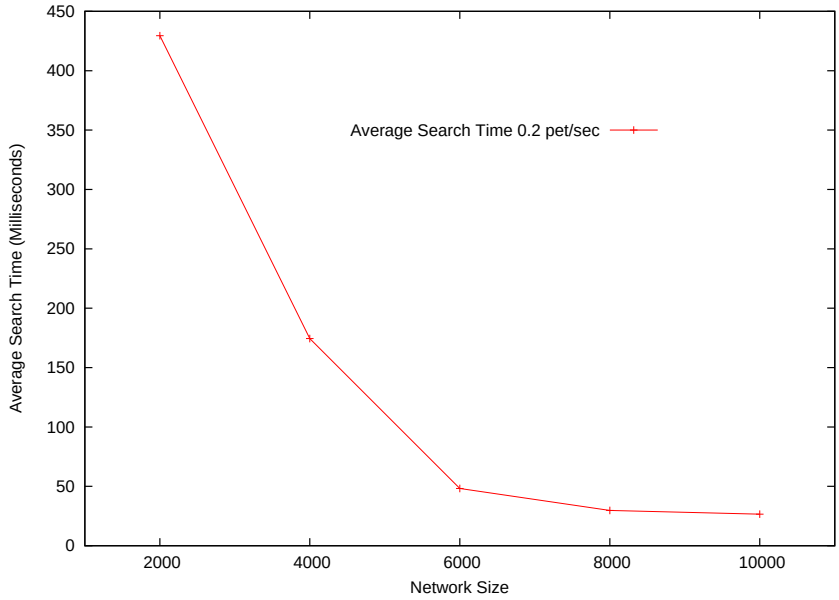


Figure 7.3: Search times as network grows.

10000 nodes. Only results related to searches started between the minutes 31 and 60 (included) of virtual time are used (about 3600000 searches). When all searches started before or at the minute 60 are over the simulation is stopped. The replication rate used is $r = 0.05$, the time between searches is $tbs = 5$ seconds.

Observing Figure 7.3 we detect a somewhat surprising effect: the average search time decreases as the network grows. This is due to the fact that, the bigger the network is, following the proportions depicted on Table 7.1 the more high capable nodes are added to the system. This leads to a better performance as those nodes can manage many connections and so decrease the mean number of hops needed to solve searches, as is plotted in Figure 7.4.

It seems then that we can conclude that DANTE is able to manage properly big network sizes, as long as there is a certain proportion of high capacity nodes present in the system.

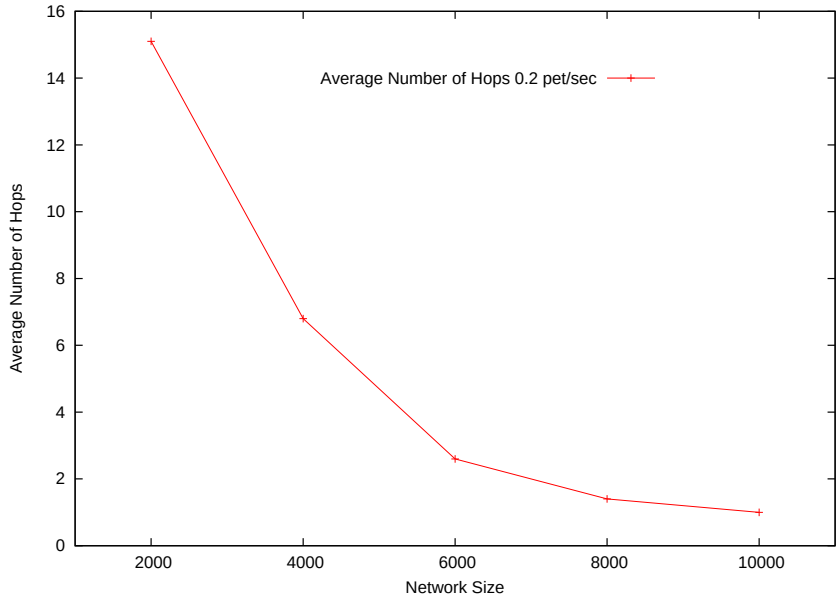


Figure 7.4: Number of hops as network grows.

7.5.3 Robustness Against Peers Churn

It is well known that peers may enter and leave the network at a high rate. This can, in some cases, compromise the efficiency of the system. In this section we present some simulations results that show how DANTE performance is not strongly affected by the churn of peers.

The simulations of this section are run with 10000 nodes. The replication is $r = 0.05$ and the time between searches is $tbs = 5$ seconds. Only searches started between minutes 31 and 60 (included) are taken into account for the results. Searches started before minute 31 are discarded to avoid the initial transition state. The simulations were run until all searches started before minute 61 were finished.

Initially, nodes are active with a probability of 0.5. At start time, active nodes form a random topology. Each active node will run for a certain time that is calculated (a different time for each one) using a exponential distribution. When the time expires, the node is deactivated: it discards all searches in its

queue, and closes its connections with all its neighbors. After 0.5 seconds of virtual time, the node changes to the active state again, pointing its *native* connections to 10 nodes chosen at random, and the time to remain active is recalculated. This is similar to simulate nodes leaving the system as other nodes simultaneously join it (a similar approach is used in [14]).

To simulate different churns, we set different values for the mean of the exponential distribution used to compute the time nodes will stay active: 1, 5, 10, 50 and 100 minutes. We deem that the means 1, 5 and 10 minutes simulate a churn somewhat higher than what is usually found in real world networks, nonetheless we use them to measure the performance of DANTE even under those extreme circumstances.

Finished searches are classified into three categories: *successful*, *failed* (the search TTL expired before the resource was found) or *discarded* (search was in the queue or processed by a node that changed to the deactivated state). Figure 7.5 shows searches results for each experiment. For the higher churn, the number of discarded plus failed searches is about 10% of the total. Yet, when the mean of the distribution increases, the number of discarded and failed searches decreases sharply. When the mean is 10 minutes, the number of not successful searches (approximately 2600) represent only the 0.07% of the total.

In Figures 7.6 and 7.7 it is shown how the churn of peers affects the search performance (only for successful searches). Although the number of jumps and the time to complete queries increase, they are within what we deem are acceptable bounds even when the churn of peers is high.

7.5.4 Robustness Against Attacks

In DANTE, high capacity nodes tend to have more connections, even forming a starlike topology. Thus, it can be argued that DANTE is vulnerable to attacks targeted to well-connected nodes. In this section we discuss how attacks can affect DANTE's behavior. The simulations presented here were run with 10000 nodes, and replication $r = 0.01$ (each resource is held by only one node, so we

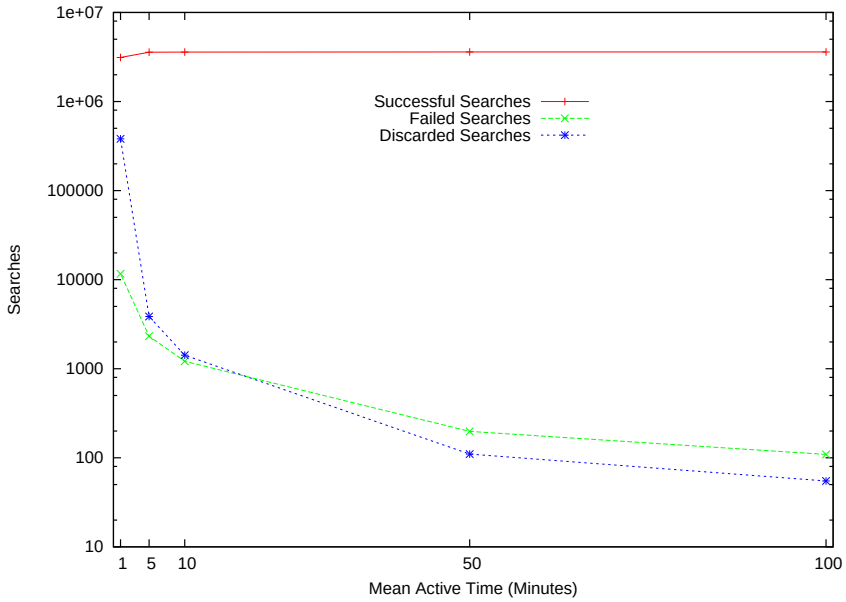


Figure 7.5: Search results under churn.

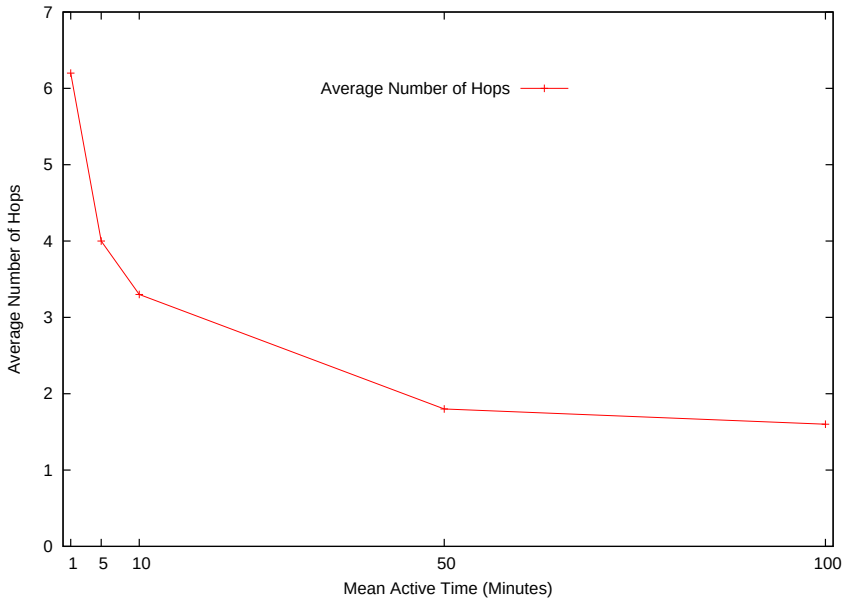


Figure 7.6: Number of hops under churn.

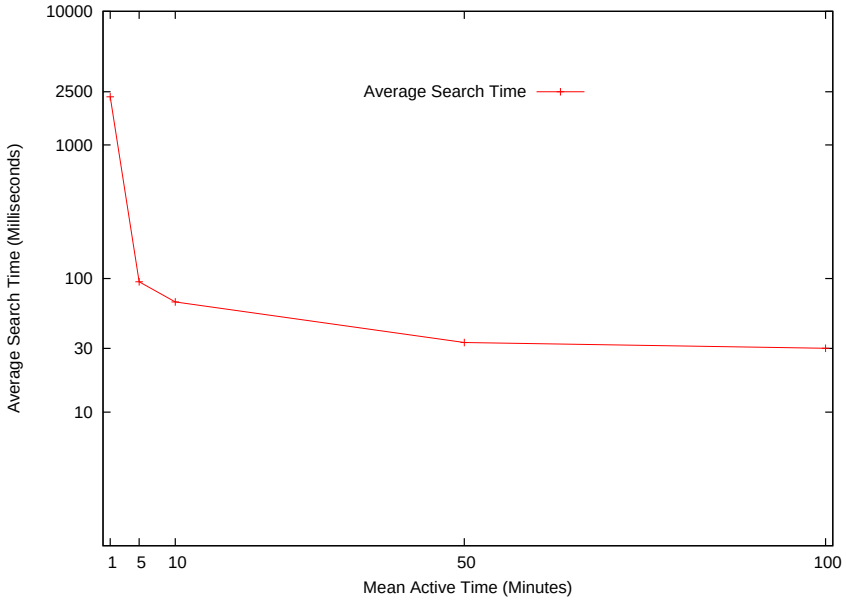


Figure 7.7: Search times under churn.

impose a tough condition here). Two different loads were tested.

The results are shown in Figures 7.8 and 7.9. Both of them show how the network behavior evolves as the virtual time passes, from the first minute of simulation (remember that initially nodes form a random topology) to minute 90. At minute 30, when the network has moved to a starlike topology, an attack is performed: the 10 best connected nodes (central nodes) are forced to leave the network. Those are also the 10 most capable nodes in the network (see Table 7.1). 30 minutes later, those nodes are reactivated. We will check how DANTE reacts to those events.

Figure 7.8 shows how the average number of hops to find resources decreases sharply in a few reconnections until it reaches a value close to 1. At that moment the network has a starlike topology. When the attack is performed at minute 30, the remaining nodes redirect their connections randomly, so a random topology appears again. From that moment on, nodes will try to connect to the remaining peers with higher capacity (in this case, nodes of the fourth Capacity Level at Table 7.1). The network is never centralized again, as there are no nodes with

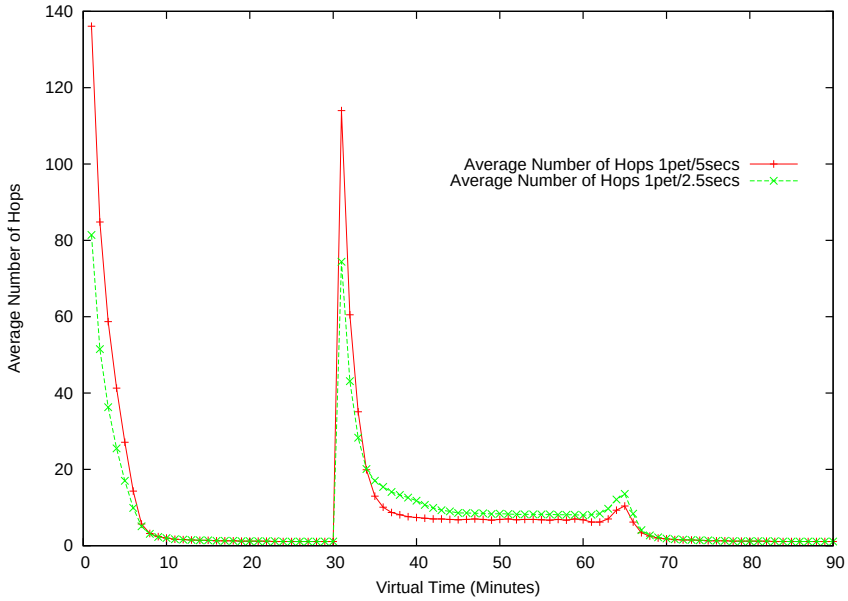


Figure 7.8: Number of hops under attack.

capacity enough to become central. Yet, highly connected nodes appear so the mean number of hops decreases sharply in few minutes (to lesser values with lesser load). Finally, when the attacked nodes are back, the network changes again to a starlike topology.

In Figure 7.9 we see how the attack affects to the search times. As expected, those times increase to values close to those obtained at the beginning of the experiment. Then, as nodes change their connections the topology is adapted again, lowering the average search time to a fair value. Finally, when the 10 nodes attacked are back, the search times gradually return to the values previous to the attack.

The proportion of discarded searches is around the 0.005% of the total in both experiments. The proportion of failed searches is less than the 0.04%, again in both experiments. Please note that a certain number of failed searches could not be avoided in these simulations, as each resource was held by only one node, and the attacked peers are not present for 30 minutes. Then, searches run during that interval for resources in those nodes could not finish successfully.

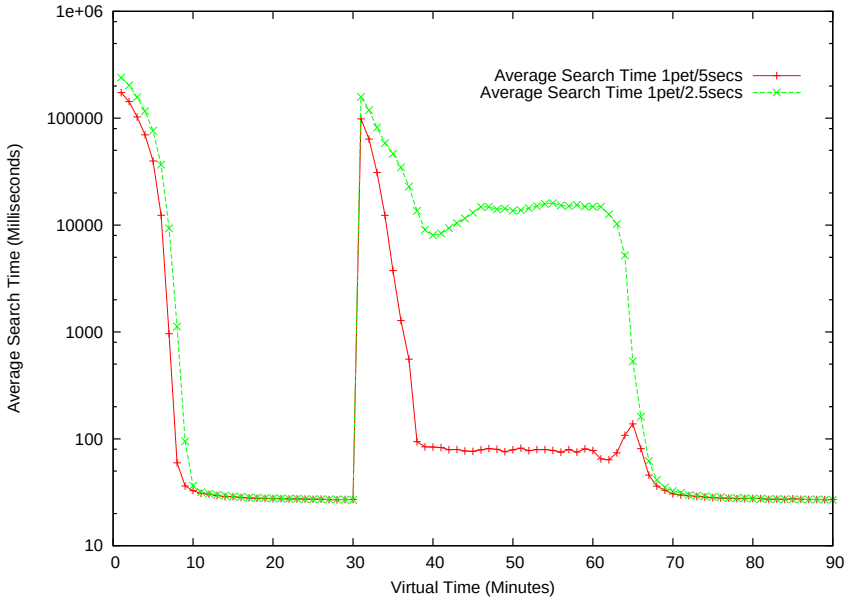


Figure 7.9: Search times under attack.

We can conclude that DANTE can be temporarily affected by well targeted attacks. How important is this effect depends on factors like which are the attacked nodes or the load in the network. Yet, even in a scenario where nodes that have become central are all successfully attacked at the same time, and no other nodes of the same capacity remain in the system, the network adapts again to reach another efficient state. The system is never fully shut down, because it is not dependant on any particular subset of nodes.

7.5.5 DANTE vs. GIA

As explained in section 3.3.2, Gia is another proposal of a P2P system that uses an adaptation mechanism to improve the efficiency of searches. In [14] Gia authors carried on some simulations that show how self-adapting networks can offer a better performance than other solutions as flooding in a variety of scenarios. Thus, instead of repeating those same simulations with DANTE, we have deemed more interesting to compare Gia and DANTE.

We have developed a Gia simulator that implements the mechanisms de-

scribed in [14]: a *flow control* system to avoid overloading nodes, a *biased random walk* search mechanism, and a *topology adaptation* protocol.

An important parameter of Gia is the *maximum number of neighbors* (max_neigh). Nodes in Gia try to connect to as many nodes as possible, and to those with the highest capacity. We have set that limit to 20 (twice the number of *native* connections in DANTE), so that the average degree is the same as the one obtained in the DANTE simulations. Gia advocates could reason that setting a higher maximum bound would improve performance, as searches would need less hops to locate resources. But then, it would be enough in DANTE to increase the number of nodes *native* connections to $max\_neigh/2$ again.

Simulations were run with 1000 nodes, with a replication $r = 0.1$. As usual, nodes capacities and bandwidth are set following the distribution of Table 7.1. Only searches started between minutes 31 and 60 are taken into account.

In Figure 7.10 we plot the average search times for different loads on both systems. DANTE seems to perform better than Gia for all loads. Additionally, beyond a certain point, Gia search times start to grow quickly with the system load, while DANTE is able to keep search times low for the same loads. Figure 7.11 helps us to understand the reason of DANTE's better behavior: searches in Gia need many more jumps to find a certain resource (about 160) than in DANTE (about 7). The reason is that, although the topology in DANTE is not totally centralized (as there are not enough high capacity nodes), it still keeps a clustered form where a few nodes are well connected and hence allows queries to be completed in a few hops. Gia, on the other hand, is unable to form such a topology. With certain loads, this leads to a low system performance.

All searches were successfully finished in DANTE. Gia, on the other hand, presented a certain proportion of failed searches (between 1.5% and 2%) in all experiments.

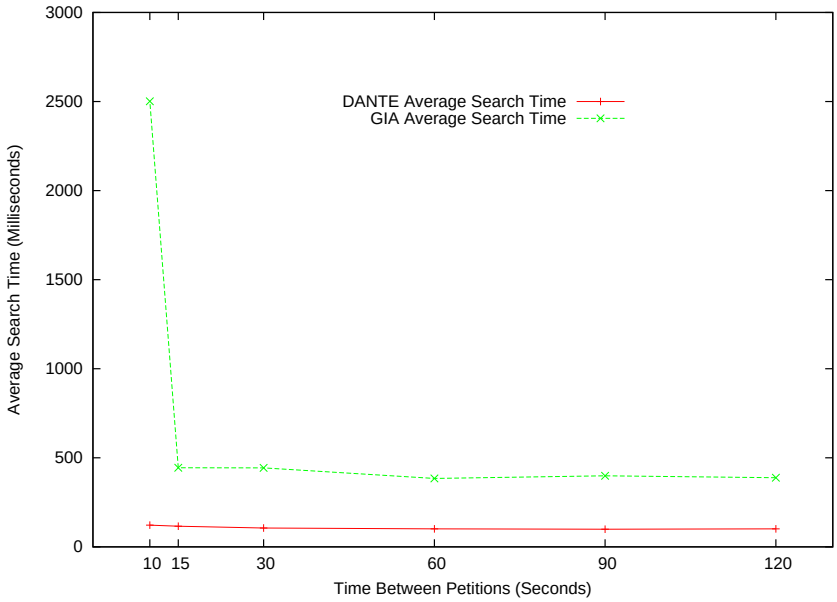


Figure 7.10: Gia and DANTE search times.

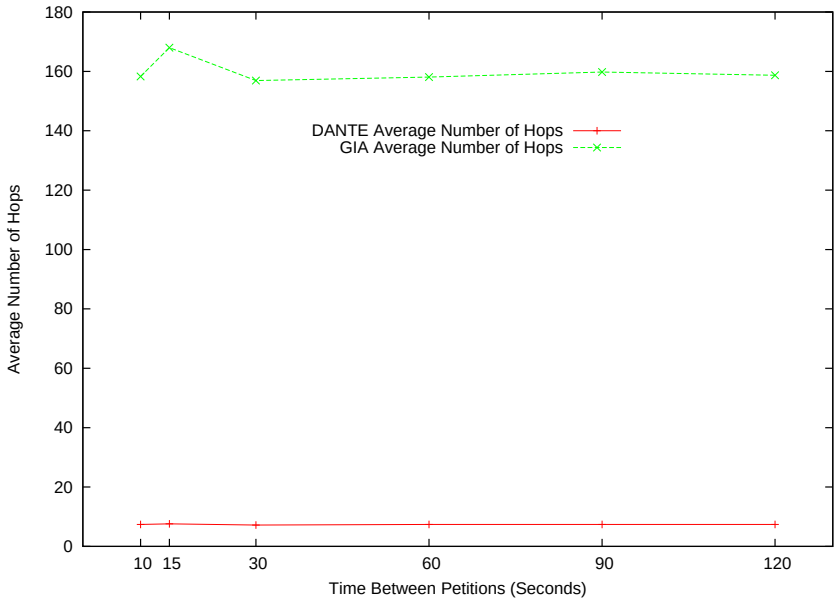


Figure 7.11: Gia and DANTE number of hops.

7.6 Reconnection Kernel Design

When creating DANTE, several questions arise. First of all, how to design a kernel, able to take into account nodes' load and heterogeneity. We developed and tested different candidates.

In this section we show the results of the experiments performed, that justify the design choices taken when developing DANTE.

7.6.1 Comparing DANTE Kernel Candidates

The key component of DANTE's design is the *attachment kernel*, that computes the attractiveness of candidate peers in the reconnection process. The kernel decides to which other peers the node executing the reconnection must try to connect to. So, at the end, the topology evolution of the overall system is decided by the kernel. A proper kernel design is mandatory to make the topology adapt as intended.

We present here the kernel candidates that were conceived. All of them are based in Cholvi's idea of computing the attraction of a peer as a power of the peer's degree. The difference is in the way the exponent is computed for each proposal.

Kernel based on average searches service times

In this kernel, the attractiveness depends on the average time needed to service each search, that is, the time to process it plus the time the search must wait in queue. Let C be the set of candidates, T_{si} the average service time and k_i the degree of each node $i \in C$. Let $T_{s_{\min}}$ the minimum average service time, $T_{s_{\min}} = \min_{i \in C}\{T_{si}\}$. Then the attractiveness of i, Π_i, is computed as:

$$\Pi_i = k_i^{\frac{2}{T_{si}/T_{s_{\min}}}} \tag{7.6}$$

as you see, this formula has the same form than the one proposed by Cholvi *et al.*, described in section 4.2. But the exponent is changed so it has greater

values as the average search time decreases. Only the peer with the minimum average search time will reach an exponent of 2.

Kernel based on load

This kernel is more complex than the previous one. It uses the node's load and throughput to compute the *congestion* of that node. The congestion is then used to decrease the value of the exponent, which has a value between 2 and 0. Let λ_i and μ_i the load and throughput of node $i \in C$, where C is the list of candidates. Then, congestion of node i, g_i, is computed as:

$$g_i = \frac{\lambda_i}{\mu_i} \tag{7.7}$$

The more congested the peer is, the lower the exponent should be. Thus, we express the attractiveness as:

$$\Pi_i = k_i^{2 \times (1 - g_i)} \tag{7.8}$$

We found out that is desirable to 'tune' the congestion impact on the exponent, so a small congestion has little effect, but as the congestion grows the impact also grows steadily. The goal is to avoid that with little congestion powerful nodes get discarded as good candidates. To 'tune' the congestion, it is raised to an exponent *alpha*, where *alpha* is a parameter of the experiment. So finally the attractiveness of node i is computed using the expression:

$$\Pi_i = k_i^{2 \times (1 - g_i^{\alpha})} \tag{7.9}$$

Kernel based on capacities and average service times

It is the one finally used in DANTE, and it is explained in section 7.2.

Simulation results

To compare the three kernels, we run a series of experiments. The configuration was set as follows:

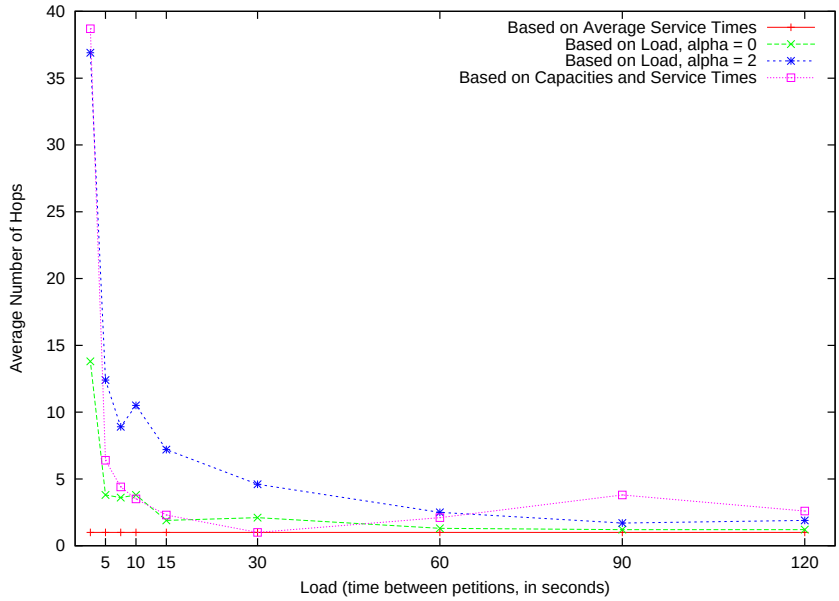


Figure 7.12: Average number of hops for different attachment kernels.

- 10000 nodes, with the distribution of capacities and bandwidths depicted in Table 7.1.

- Resources replication $r = 0.01$. Each resource is held by only one node. Each peer holds 100 resources. All resources are equally popular.

- TTL of *Look For Node* messages was set to 30. TTL of resource search messages was set to 1000.

- Nodes handle 10 *native* connections each. Reconnections are triggered every 30 seconds of virtual time. Nodes change 5 *native* connections at each reconnection.

Different loads were tested for each kernel. Load was set by the *time between searches*, *tbs*, parameter. 7 values were tested: 120, 90, 60, 30, 15, 7.5, 10, 5 and 2.5 seconds between petitions. In Figure 7.12 we show the average number of hops needed to find a resource.

In Figure 7.12 we observe that for low loads all kernels achieve a centralized, or closed to centralized, topology, as the average number of hops is close to 1. But as the load is increased, the number of hops grows for all kernels, save the one based on capacities and average service times. The conclusion is that this kernel is very resilient to abandon the centralized topology even if the load grows. We deem this is a good property, because, as we said before, DANTE tries to achieve centralized topologies as long as nodes do not become congested and our experiments show that central nodes can still handle all the incoming traffic even for the greatest load.

Thus, the kernel using the nodes capacities and average service times seems to be more stable, while at the same time keeps the property of adaptability to high loads to avoid congestion as we saw in the experiments shown in section 7.5. Because of this, we decided to use it as the *attachment kernel* in DANTE.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis is part of the efforts that the research community has undertaken to improve the efficiency of P2P networks, and in particular the resource location process on these systems.

First, we have depicted the present solutions for resource location, their advantages and drawbacks. We have also explained how, due to the limitations on structured networks, certain research is focused on further improving search mechanism in unstructured networks. As we have seen, proposals based on random walks and dynamic, self-adapting topologies have lead so far to some good results.

The main goal of our work has been to contribute to the P2P research field, by introducing a model that helps to understand how random walks behave in P2P networks, and proposing a new P2P self-adapting system.

Our experiments with random (Erdos-Renyi), and power-law (Newman and Barabasi) networks prove that the model can provide good average estimations of two important magnitudes: the amount of different peers *visited* and *known* at each hop of a random walk. These estimations are computed using only the degrees distribution of the network, as a way to characterize the network topology. Also by simulations we have checked how our estimation of the *average search length* is close to the real results. The greatest error, obtained for Barabasi

networks, was generally close to the 10%. But, maybe more important is another fact also probed by our simulations: **neither the number of unknown nodes nor the average search length can be computed using only the degrees distribution**. As we have seen, similar degrees distribution lead to different average search length depending on how the network is built. The results for networks built using the Barabasi approach are different to those of networks built with the Newman proposal, even when the degrees distributions are similar. Yet, despite this, from the simulations we infer that our metric is close enough to be representative of the real searches length.

With the second part of our model we have intended to deliver a set of expressions, based on network queuing theory, that allow to compute the average time for searches on a P2P system. The degree and processing capacity of each node were the only data required. As these expressions use as a parameter the expected average number of hops of searches, that is computed using the first part of the model, a certain inaccuracy could be expected. Nonetheless, the simulations we run confirm the goodness of this model, save for Barabasi networks.

On the other hand DANTE, our proposal of a P2P system with dynamic topologies, has met the requirements that we stated at the beginning. Our implementation of Cholvi's system allowed to confirm that trying to form centralized or random topologies for low and high loads, respectively, indeed worked and behave as predicted. It also helped to design DANTE's reconnection mechanism, once the potential problems of Cholvi's proposal were detected. In our simulations, DANTE successfully adapts an heterogeneous system to changing load conditions. It also reacts properly to attacks, and handles well a high transitivity of peers.

8.2 Future Work

The main goals stated at the beginning of this research have been met. Yet, open issues have appeared, that could lead to better results and could deserve

further efforts.

- New kernels for DANTE can be proposed and studied. The reconnection kernel presented in this thesis as the basis of DANTE seems to work quite well, as our simulations prove. However, we are confident that even better reconnection kernels can be designed. Kernels that achieve better topologies, or make the system to evolve faster. In any case, those kernels should follow the same goals than the actual one: to form highly centralized topologies where more capable nodes get more connections, preventing at the same time that any peer gets overloaded.

- DANTE uses a simple, pure random walk approach. Yet, there are a number of proposals that try to improve the efficiency of random walks (see Sec. 3.2) that could be used along with DANTE. For example:

  - Using a *record of previous searches*. Each peer could keep a record of the searches that have visited it, and where those searches were forwarded. If the same search message visits again the peer, then it will be forwarded to some of the neighbors it was not forwarded to before.

  - Storing in each search message the list of nodes that it has traversed before. When forwarding a message, the node must try to avoid sending it to an already visited peer.

  - Applying solutions like *Adaptative Probabilistic Search* or *Routing Index*, see Sec. 3.2.1.

- Resources *caches*. When nodes disconnect, they do not keep the list of resources of their neighbors. If nodes have storing capacity, there is no real reason to lose this knowledge. So, nodes could hold a cache to store the lists of resources of recent disconnected nodes. This cache should be updated periodically to delete old lists of resources.

- *Bloom filters.* A bloom filter is a compact data structure to store sets. Then, if resources have an associated set of keywords, they can be coded with bloom filters using those sets. Indeed, a bloom filter can code the entire set of documents of one node. Using bloom filters would improve overall performance because:
  - Connections setup would require less bandwidth. Instead of sending the entire list of resources to new neighbors, nodes would only send the bloom filter corresponding to their set of documents.
  - Queries would be processed faster. Each request would be checked only against the bloom filters associated with node neighbors. This would be faster that comparing the query with all known resources.
  - Caches would need less storage space.
  - So far, DANTE uses one-hop replication: each peer knows the resources stored by its neighbors, but not beyond (resources held by the neighbors of their neighbors, etc). This is mainly due to the high amounts of bandwidth and storage capacity that would be required. However, applying bloom filters could make less resource demanding, and so feasible, using two-hops replication.

Finally, future work regarding the mathematical model presented in this thesis could also be undertaken. Further research concerning the validity of the model for other networks could be done. Moreover, it is possible that, using as a basis the part of the model that computes the average searches time, new expressions can be found. Those results could afterwards lead to a better understanding of how the efficiency is related with the degree distribution, and so new heuristics for topology adaptation could be inferred.

Bibliography

[1] The extensible messaging and presence protocol (xmpp) rfc. http://www.xmpp.org/rfcs.

[2] The jabber website. http://www.jabber.com.

[3] The setiathome project website. http://setiathome.ssl.berkely.edu.

[4] Abello, J., Buchsbaum, A., and Westbrook, J. A functional approach to external graph algorithms. In *Lecture Notes in Computer Science, Proceedings of the 6th Annual European Symposium on Algorithms (ESA '98)* (1998), Springer-Verlag, pp. 332–343.

[5] Adamic, L. A., Huberman, B. A., Lukose, R. M., and Puniyani, A. R. Search in power law networks. *Physical Review E 64* (October 2001), 46135–46143.

[6] Aiello, W., Chung, F., and Lu, L. A random graph model for massive graphs. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing (STOC 2000)* (2000), ACM Press, pp. 171–180.

[7] Albert, R., Yeong, H., and Barabási, A.-L. The diameter of the world wide web. *Nature 401* (1999), 130–136.

[8] Androutsellis-Theotokis, S., and Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys 36*, 4 (December 2004), 335–371.

[9] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 286* (1999), 509,512.

[10] Bisnik, N., and Abouzeid, A. Modeling and analysis of random walk search algorithms in p2p networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P 2005)* (July 2005), IEEE Computer Society, pp. 95–103.

[11] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13* (July 1970), 422–426.

[12] Bollobás, B. *Random Graphs*, 2nd ed. Cambridge University Press, 2001.

[13] Broder, A., and Mitzenmacher, M. Network applications of bloom filters: A survey, 2003.

[14] Chawathe, Y., Ratnasamy, S., Lanham, N., and Shenker, S. Making Gnutella-like P2P systems scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2003)* (Karlsruhe, Germany, August 2003), pp. 407–418.

[15] Chen, Y., Katz, R. H., and Kubiatowicz, J. Scan: A dynamic, scalable, and efficient content distribution network. In *Lecture Notes in Computing Science (Proceedings of the First International Conference on Pervasive Computing, Pervasive 2002)* (2002), vol. 2414, Springer-Verlag, pp. 282–296.

[16] Cholvi, V., Felber, P. A., and Biersack, E. W. Efficient search in unstructured peer-to-peer networks. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures* (Barcelona, Spain, June 2004), pp. 271–272.

[17] Cholvi, V., Laderas, V., López, L., and Fernández, A. Self-adapting network topologies in congested scenarios. *Physical Review E 71*, 3 (2005), 035103.

[18] Clarke, I., Miller, S. G., Hong, T. W., Sandberg, O., and Wiley, B. Protecting free expression online with freenet. *IEEE Internet Computing 6* (Jan-Feb 2002), 40–49.

[19] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. 46–66.

[20] Cohen, E., and Chenker, S. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2002)* (Pittsburgh, Pennsylvania, United States, August 2002), pp. 177–190.

[21] Cooper, B. F., and Garcia-Molina, H. Ad hoc, self-supervising peer-to-peer search networks. *ACM Transactions on Computer Systems 23* (April 2005), 169–200.

[22] Crespo, A., and Garcia-Molina, H. Routing indices for peer-to-peer systems. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems* (Vienna, Austria, July 2002), pp. 5–14.

[23] Dill, S., Kumar, S. R., McCurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. Self-similarity in the web. In *Proceedings of the 27th International Conference on Very Large Data Bases* (2001), ACM Press, pp. 69–78.

[24] Dingledine, R., Freedman, M. J., and Molnar, D. The free haven project: distributed anonymous storage service. In *Proceedings of the International Eorkshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability* (2001), pp. 67–95.

[25] Druschel, P., and Rowstron, A. I. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS 2001)* (Elmau/Oberbayern, Germany, May 2001), IEEE Computer Society, pp. 75–80.

[26] Fall, K., and Varadhan, K. The ns manual. http://www.isi.edu/nsnam/ns/doc. UC Berkeley and Xerox PARC.

[27] Faloutsos, M., Faloutsos, P., and Faloutsos, C. On power-law relationships of the internet topology. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (1999), ACM Press, pp. 251–262.

[28] Ferreira, R. A., Ramanathan, M. K., Grama, A., and Jagannathan, S. Efficient randomized search algorithms in unstructured peer-to-peer networks.

[29] Fletcher, G. H. L., Sheth, H. A., and Borner, K. Unstructured peer-to-peer networks: Topological properties and search performance. In *Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing* (New York, New York, United States, July 2004). To be published by Springer.

[30] Foster, I., and Iamnitchi, A. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Lecture Notes in Computer Science, Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03)* (2003), vol. 2735, Springer-Verlag, pp. 118–128.

[31] Gkantsidis, C., Mihail, M., and Saberi, A. Random walks in peer-to-peer networks. In *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004* (Hong Kong, March 2004), vol. 1, pp. 120–130.

[32] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I. Load balancing in dynamic structured P2P systems. In *Proceedings of the 2004 Conference on Computer Communications, Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)* (2004), vol. 4, pp. 2253–2262.

[33] Goyal, P., Vin, H. M., and Cheng, H. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *Proceedings of the 1996 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 1996)* (Palo Alto, California, United States, 1996), pp. 157–168.

[34] Guimerà, R., Díaz-Guilera, A., Vega-Redondo, F., Cabrales, A., and Arenas, A. Optimal network topologies for local search with congestion. *Physical Review Letters 89* (November 2002).

[35] Gummadi, K. P., Dunn, R. J., Saroiu, S., nad Henry M. Levy, S. D. G., and Zahorjan, J. Measurement, modeling and analyis of a peer-to-peer file-sharing workload. In *Proceedings of the 2003 ACM Symposium on Operating Systems Principles (SOSP 2003)* (Bolton Landing, New York, 2003), pp. 314–329.

[36] Hand, S., and Roscoe, T. Mnemosyne: Peer-to-peer steganographic storage. In *Lecture Notes in Computer Science, Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)* (2002), vol. 2429, Springer-Verlag, pp. 130–140.

[37] Harvey, N. J. A., Jones, M. B., Saroiu, S., Theimer, M., and Wolman, A. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), in electronic format at http://www.usenix.org/publications* (Seattle, United States, March 2003).

[38] Iamnitchi, A., and Foster, I. *A Peer-to-Peer Approach to Resource Location in Grid Environments*. IEEE Computer Society, 2004, pp. 413–429.

[39] Jelasity, M., Guerraoui, R., Kermarrec, A.-M., and van Steen, M. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Lecture Notes in Computer Science (Proceedings of Middleware 2004)* (October 2004), vol. 3231, Springer-Verlag, pp. 79–98.

[40] Jovanovic, M. A., Annexstein, F. S., and Berman, K. A. Modeling peer-to-peer network topologies through 'small-world' models and power laws. In *Proceedings of the IX. Telecommunications Forum (TELFOR 2001)* (2001).

[41] Jovanovic, M. A., Annexstein, F. S., and Berman, K. A. Scalability issues in large peer-to-peer networks - a case study of gnutella, 2001.

[42] Kalogeraki, V., Gunopulos, D., and Zeinalipour-Yazti, D. A local search mechanism for peer-to-peer networks. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management* (McLean, Virginia, United States, November 2002), pp. 300–307.

[43] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing (STOC'97)* (1997), pp. 654,663.

[44] Keromytis, A. D., Misra, V., and Rubenstein, D. Sos: secure overlay services. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2002), ACM Press, pp. 61–72.

[45] Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. S. The web as a graph: Measurements, models and methods. In *Lecture Notes in Computer Science, Proceedings of Computing and Combinatorics: Fifth Annual International Conference (COCOON'99)* (1999), vol. 1627, Springer-Verlag, pp. 1–17.

[46] Krapivsky, P. L., Redner, S., and Leyvraz, F. Connectivity of growing random networks. *Physical Review Letters 85* (November 2000), 4629–4632.

[47] Kretschmar, M., and Morris, M. Measures of concurrency in networks and the spread of infectious disease. 165–195.

[48] Krishnamurthy, B., Wang, J., and Xie, Y. Early measurements of a cluster-based architecture for p2p systems. In *Proocedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (San Francisco, California, USA, November 2001), ACM Press, pp. 105–109.

[49] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ACM ASPLOS 2000)* (Massachusetts, United States, November 2000), pp. 129–138.

[50] Kulbak, Y., and Bickson, D. *The eMule Protocol Specification, http://www.cs.huji.ac.il/labs/danss/presentations/emule.pdf*, 2005.

[51] Liang, J., Kumar, R., and Ross, K. W. Understanding kazaa (submitted).

[52] Liang, J., Kumar, R., and Ross, K. W. The fasttrack overlay: a measurement study. *Computer Networks: The International Journal of Computer and Telecommunications Networking 50* (April 2006), 842–858.

[53] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing* (New York, New York, United States, June 2005), pp. 84–95.

[54] Lv, Q., Ratnasamy, S., and Shenker, S. Can heterogeneity make Gnutella scalable? In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (Cambridge, United States, March 2002), pp. 94–103.

[55] Malkhi, D., Naor, M., and Ratajczak, D. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC 2002)* (2002), ACM Press, pp. 183–192.

[56] Maymounkov, P., and Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the First International Workshop on Peer-to-Peer Systems* (Cambride, United States, March 2002), pp. 53–65.

[57] Menasce, D. A., and Kanchanapalli, L. Probabilistic scalable p2p resource location services. *ACM SIGMETRICS Performance Evaluation Review 30*, 2 (September 2002), 48–58.

[58] Newman, M. E. J. A measure of betweenness centrality based on random walks. *Social Networks 27* (2005), 39–54.

[59] Newman, M. E. J., Strogatz, S. H., and Watts, D. J. Random graphs with arbitrary degree distributions and their applications. *Physical Review E 64*, 2 (Jul 2001), 026118–1,026118–17.

[60] Plaxton, C. G., Rajaraman, R., and Richa, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)* (June 1997), pp. 311–320.

[61] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)* (San Diego, California, United States, 2001), pp. 161–1672.

[62] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S. Application-level multicast using content-addressable networks. In *Lecture Notes in Computer Science, Proceedings of the Third International COST264 Workshop on Networked Group Communication* (2001), vol. 2233, Springer-Verlag, pp. 14–29.

[63] Reynolds, P., and Vahdat, A. Efficient peer-to-peer keyword searching. In *Lecture Notes in Computer Science (Proceedings of Middleware 2003)* (2003), vol. 2672, Springer-Verlag, pp. 21–40.

[64] Rhea, S. C., and Kubiatowicz, J. Probabilistic location and routing. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM 2002)* (New York, USA, June 2002), vol. 3, IEEE Computer Society, pp. 1248– 1257.

[65] Ripeanu, M., Lamnitchi, A., and Foster, I. Mapping the gnutella network. 50–57.

[66] Ritter, J. Why gnutella can't scale. no, really. Tech. rep. Electronic format in http://www.darkridge.com/ jpr5/doc/gnutella.html.

[67] Rowstron, A. I. T., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms* (Heidelberg, Germany, 2001), pp. 329–350.

[68] Saroiu, S., Gummadi, P. K., and Gribble, S. D. A measurement study of peer-to-peer file sharing systems. In *Proceedings of SPIE (Proceedings of Multimedia Computing and Networking 2002, MMCN'02)* (2002), vol. 4673, pp. 156–170.

[69] Sen, S., and Wang, J. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the Second Annual ACM Internet Measurement Workshop* (November 2002).

[70] Silvey, P., and Hurwitz, L. Adapting peer-to-peer topologies to improve system performance. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)* (January 2004), vol. 7, IEEE Computer Society, p. 70199.1.

[71] Sripanidkulchai, K. The popularity of gnutella queries and its implications on scalability. In O'Reilly's P2P Open Conference, http//www.openp2p.com.

[72] Stallings, W. *Queuing Analysis*. Available on line, ftp://shell.shore.net/members/w/s/ws/Support/QueuingAnalysis.pdf, 2000.

[73] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)* (San Diego, CA, United States, 2001), pp. 149–160.

[74] Strogatz, S. H., and Watts, D. J. Collective dynamics of small-world networks. *Nature 393*, 6684 (1998), 409–410.

[75] Tsoumakos, D., and Roussopoulos, N. Adaptive probabilistic search for peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing* (September 2003), pp. 102–109.

[76] Tsoumakos, D., and Roussopoulos, N. Analysis and comparison of p2p search methods. In *ACM International Conference Proceedings Series, Proceedings of the 1st International Conference on Scalable Information Systems (Infoscale 2006)* (2006), vol. 152, ACM Press.

[77] The gnutella website. http://www.gnutella.com.

[78] The kazaa website. http://www.kazaa.com.

[79] The napster website. http://www.napster.com.

[80] The edonkey website. http://www.edonkey2000.com.

[81] The overnet website. http://www.overnet.com.

[82] Jackson's theorem, from wikipedia. Available on line, http://en.wikipedia.org/wiki/Jackson_theorem.

[83] Sha hash functions, from wikipedia. Available on line, http://en.wikipedia.org/wiki/SHA_hash_functions.

[84] Wilf, H. S. *Generatingfunctionology.* Academic Press, 1994.

[85] Yang, B., and Garcia-Molina, H. Comparing hybrid peer-to-peer systems. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB Conference)* (Rome, Italy, September 2001), pp. 561–570.

[86] Yang, B., and Garcia-Molina, H. Improving search in peer-to-peer networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems* (Vienna, Austria, July 2002), pp. 5–14.

[87] Yang, B., and Garcia-Molina, H. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering* (Bangalore, India, March 2003), IEEE Computer Society, pp. 49–60.

[88] YING TYAN, H. *Design, Realization, and Evaluation of a Component-based Compositional Software Architecture for Network Simulation.* PhD thesis, The Ohio State University, 2002.

[89] ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. rep., University of California, Berkeley, 2001.

Appendix A

Resumen en español

En este apéndice se incluye un resumen extendido en castellano de los contenidos de esta tesis doctoral[1].

A.1 Antecedentes

Esta tesis presenta el trabajo de investigación del autor en el campo de las redes *entre iguales* (*peer-to-peer*, P2P).

Los sistemas P2P han ganado una gran popularidad en los últimos años. Son un nuevo paradigma de comunicación en el que los recursos (tales como ficheros, servicios, etc), son a la vez demandados y proporcionados por los miembros del sistema (*pares* de la red). Esta aproximación contrasta con la arquitectura tradicional cliente-servidor, donde cada participante de la red tiene un papel bien definido. En los sistemas P2P, sin embargo, cada par puede ser a la vez un cliente que demanda recursos y un servidor que los proporciona.

En comparación con los sistemas tradicionales, los sistemas P2P presentan ventajas como una mayor flexibilidad, mayor tolerancia a fallos y mejor escalabilidad. Estas ventajas se deben a que no existe una entidad central de la que dependa el funcionamiento del sistema. Por otro lado, esta misma falta de

[1]De acuerdo con los requisitos establecidos por el Punto Quinto (Presentación y Depósito de Tesis Doctoral) de los Procedimientos a seguir según la Normativa Reguladora del Tercer Ciclo de la Universidad Rey Juan Carlos.

servicios centralizados hace que ciertas tareas, como la localización de recursos, se vea dificultada ya que no existe un índice central de los recursos disponibles al que poder realizar consultas.

La comunidad académica ha propuesto varias soluciones para resolver el problema de la localización eficiente de recursos [8]. Estos trabajos se basan en construir una red *superpuesta* (*overlay network*) sobre la red física. Los participantes del sistema P2P se encuentran conectados a través de enlaces en la red superpuesta, formando una topología (denominada a veces *overlay topology*). Los *vecinos* de un nodo dado son aquellos pares con los que se encuentra conectado en la red superpuesta.

Según cómo se construye la red superpuesta, las soluciones para la búsqueda de recursos se agrupan en dos grandes familias, redes *estructuradas* y redes *no estructuradas*.

Debido a cómo están construidas, las redes estructuradas garantizan encontrar el recurso buscado en pocos saltos ($O(\log(N))$, donde N es el tamaño de la red), lo que implica que cada búsqueda consumirá muy pocos recursos. Además, en estos sistemas no se da el problema de los *falsos negativos*. Un falso negativo se produce cuando una búsqueda termina sin haber encontrado el recurso, aunque el recurso sí se encuentre en la red. Las redes estructuradas aseguran que, si el recurso está en la red, será encontrado por la búsqueda.

Las redes no estructuradas son comparativamente menos eficientes, y no garantizan que no se produzcan falsos negativos. Pero por otro lado se ajustan mejor a situaciones donde la entrada y salida de nodos del sistema es frecuente, lo que es típico en muchas redes reales, como por ejemplo las redes P2P para la distribución de contenido audiovisual. La razón es que en los sistemas estructurados cada vez que entra o sale un participante de la red se debe de producir un reajuste de la topología, ya que está fijado, para cada nodo, qué otros pares deben ser sus vecinos según los nodos presentes en el sistema. Este reajuste es un proceso costoso. Las redes no estructuradas, por otro lado, no necesitan hacer ese reajuste ya que cualquier nodo puede ser vecino de cualquier otro nodo.

Por esta y otras razones, las redes no estructuradas pueden ser una opción más conveniente en algunos escenarios.

Entre las propuestas para la mejora del rendimiento de las redes no estructuradas [54] destacan las basadas en el uso de *caminos aleatorios* [31], un mecanismo de búsqueda que demanda muy pocos recursos. Se dice que una búsqueda sigue un camino aleatorio si, en cada salto, el nodo que reenvía la búsqueda elige el destino de manera aleatoria, con probabilidad uniforme, entre todos sus vecinos.

Para mejorar la eficiencia de los caminos aleatorios en términos de tiempos de búsqueda y tasa de éxitos se ha propuesto el uso de *topologías dinámicas* [17]. Estos trabajos parten del hecho de que la eficiencia de los caminos aleatorios depende en gran medida de la topología de la red superpuesta, y estudian cómo un sistema P2P puede construir topologías que mejoren el rendimiento de las búsquedas.

En muchos trabajos que estudian el comportamiento de los caminos aleatorios se asume que cada nodo puede responder por sus vecinos [5] [14] [54]. Esto complica el modelo de la red, pero lo hace más cercano a muchos escenarios reales (por ejemplo, una red social). En una red P2P, esta suposición significa que cada nodo ha de conocer los recursos de sus vecinos en la red superpuesta. En un escenario así, es fácil comprender que el tener un conjunto de pares bien conectados al resto de la red haría que los caminos aleatorios tuvieran éxito en un número de saltos reducido. Por otro lado, ha de evitarse el saturar a esos nodos bien conocidos para que no se conviertan en un cuello de botella del sistema.

Esta tesis se enmarca dentro de los esfuerzos de la comunidad académica para entender y aplicar las relaciones entre la topología de una red y el rendimiento de las búsquedas mediante caminos aleatorios. Presenta un modelo que estima cómo evoluciona el *conocimiento* de la red según se avanza en un camino aleatorio, lo que permite dar una predicción aproximada de la longitud media de las búsquedas. Además, propone un mecanismo de reconexión de red que permite a

un sistema P2P formar topologías eficientes, y estudia su rendimiento mediante simulaciones.

A.2 Objetivos

Los objetivos de esta tesis son dos: proporcionar un modelo que ayude a comprender mejor el comportamiento de los caminos aleatorios según la topología de la red, y proponer un sistema P2P con topología dinámica que, adaptándose a las condiciones de carga, busque la configuración que haga más eficientes las búsquedas por caminos aleatorios.

A.2.1 Modelo de caminos aleatorios en redes

Este modelo intenta dar una serie de estimadores del rendimiento de las búsquedas mediante caminos aleatorios en redes. Se asume que cada nodo conoce a sus vecinos, y que puede responder por ellos cuando llega una búsqueda. El modelo consta de dos partes.

La primera parte intenta proporcionar buenas estimaciones de un conjunto de métricas que caracterizan cómo evoluciona el *conocimiento* de la red que se acumula siguiendo un camino aleatorio. Al contrario que en otros trabajos [31], las búsquedas no se modelan como un proceso estocástico, sino que se asume que un camino aleatorio tiene un *estado* que se modifica según el número de saltos efectuados.

Para caracterizar el estado de un camino aleatorio usamos las siguientes métricas:

- Número de nodos *visitados* por el camino aleatorio.
- Número de conexiones comprobadas. Son las conexiones al otro extremo de cada enlace de un nodo visitado.
- Número de nodos *conocidos*. Un nodo es conocido si él o algún vecino suyo ha sido visitado por el camino aleatorio.

El estado de la búsqueda nos permite determinar la probabilidad de éxito en cada salto, y a partir de esta probabilidad obtener una estimación de la longitud media de las búsquedas en la red. Para el cálculo de estos valores partimos sólo del conocimiento de la distribución de grado de la red.

La segunda parte del modelo, a partir de la longitud media de las búsquedas (calculada con las expresiones anteriores) y aplicando teoría de redes de colas, calcula el *tiempo medio para la resolución de las búsquedas*. Esto permite valorar cómo es de buena una topología a partir del tiempo que tardarán las búsquedas.

A.2.2 DANTE, un sistema P2P auto-adaptativo

DANTE es una propuesta de sistema P2P con capacidad de adaptación de la topología. Nuestro objetivo a la hora de diseñar DANTE era construir un mecanismo de adaptación viable para su implementación en sistemas P2P reales. En concreto se debía cumplir que:

- Todos los nodos implementan la misma funcionalidad.
- No se requiere conocimiento global.
- No es necesario la intervención de un coordinador ni ninguna otra entidad centralizada.

El mecanismo propuesto elige, para cada nodo, a qué otros miembros de la red ese nodo debe conectarse. Los candidatos son elegidos calculando una métrica que decide cómo de atractivo es cada uno. Esta métrica tiene en cuenta tanto el grado del nodo, como su capacidad y su carga. Básicamente, los nodos de mayor grado y capacidad son más atractivos, salvo si se acercan al punto de saturación, en cuyo caso sus vecinos tenderán a desconectarse de ellos.

A.3 Metodología

A.3.1 Características del modelo

La principal dificultad a la hora de modelar caminos aleatorios es el hecho de que no puede asumirse que sean un proceso estocástico. Esto es especialmente cierto en redes en las que la distribución del grado es muy desigual, como es el caso de las redes *ley de potencias* (*power-law*). La razón es que los caminos aleatorios tienden a visitar con mayor frecuencia los nodos de mayor grado. Así, la probabilidad de revisitar nodos ya visitados en cada salto del camino aleatorio crece rápidamente, más que la proporción de nodos visitados en sí.

Si asumimos además que cada nodo puede responder por sus vecinos (para finalizar una búsqueda con éxito basta con visitar un vecino del nodo buscado), el modelo se complica. Incluso si el nodo visitado en un salto no ha sido encontrado antes por el camino aleatorio, algunos de sus vecinos con gran probabilidad ya serán conocidos, con lo que no podemos estimar la probabilidad de éxito directamente a partir del grado del nodo.

Esto es confirmado en [5], donde los autores observan que los resultados predichos analíticamente por ellos no concuerdan con los obtenidos mediante simulación. La razón, según ellos mismos exponen, es el alto número de revisitas.

El modelo presentado en esta tesis usa una aproximación diferente. Se definen varias magnitudes que expresan cuánto acerca de la red es conocido por el camino aleatorio según la va recorriendo. Después, se muestran y razonan las expresiones con las que se calculan esas magnitudes. Finalmente, a partir de estas magnitudes se aproxima la probabilidad de éxito de una búsqueda en un camino aleatorio, y usando esa probabilidad se calcula una estimación de la longitud media esperada de las búsquedas en la red.

La segunda parte del modelo, que estima el tiempo medio que tardarán las búsquedas en la red, toma como entrada el grado y capacidad de cada nodo del sistema. Esta parte se basa en aplicar el *teorema de Jackson*, un resultado bien conocido en redes de colas que nos permite modelar cada nodo como una

cola $M/M/1$. Así se puede obtener una estimación del tiempo que residirá en el nodo cada búsqueda que llegue al mismo. Para ello, es necesario saber la carga en dicho nodo, que es estimada linealmente a partir de su grado. Finalmente, a partir de todos los tiempos de residencia en cada nodo se puede obtener el tiempo de residencia medio en toda la red, aplicando para ello la *ley de Little*.

A.3.2 Validación del modelo por simulaciones

Para validar el modelo se ha recurrido a ejecutar experimentos sobre redes de distinto tipo: redes con distribución de grado aleatoria [12], y redes con distribución de grado que sigue una ley de potencias. Estas últimas fueron construidas según dos mecanismos distintos, el propuesto por Barabasi *et.al.* [9] y el propuesto por Newman *et.al.* [59].

Las simulaciones han mostrado que para los tres tipos de redes las estimaciones dadas por el modelo son muy cercanas a la realidad, incluída la estimación de la longitud media de las búsquedas y del tiempo medio para la resolución de las mismas. Sin embargo, se ha podido observar cierto error en el caso de las redes de Barabasi, de hasta el 10%.

A.3.3 Desarrollo de DANTE

El paso inicial en el desarrollo de DANTE fue la implementación de una propuesta previa [17], basada en estudios anteriores [34], de un mecanismo de adaptación de la topología para estudiar su rendimiento. Nuestros resultados confirmaron que la implementación funcionaba como se esperaba, y que los supuestos en los que se basaba [17] acerca del rendimiento de las topologías eran correctos: las topologías centralizadas dan un rendimiento óptimo para cargas bajas, pero para cargas altas las aleatorias funcionan mejor. Sin embargo también detectamos algunas limitaciones en el mecanismo de adaptación. Eso nos llevó a crear un nuevo mecanismo con las siguientes características:

- La heterogeneidad de los nodos es tenida en cuenta.

- Para medir el nivel de saturación en un nodo, no se usa un umbral fijo (como en el caso de [17]).

- Cambios bruscos en la topología son evitados. Para ello, el *atractivo* de cada nodo como candidato al que conectarse evoluciona de manera continúa. Así se evita que otros miembros cambien sus conexiones a este nodo de manera abrupta.

A.3.4 Propiedades

Las propiedades más destacables de DANTE son:

- La localización de recursos es realizada mediante mensajes de búsqueda que siguen un camino aleatorio.

- Se usa un mecanismo de adaptación de la topología que persigue estos objetivos:

  - Hacer la topología tan centralizada como sea posible.
  - Evitar la saturación de los nodos.
  - Los nodos de mayor grado deben de ser aquellos de mayor capacidad, ya que serán los que soporten una carga mayor.

La adaptación se hace mediante un *mecanismo de reconexión* que ejecutan todos los nodos del sistema. Periódicamente, se dispara dicho mecanismo, que es el encargado de elegir a qué otros pares debe de conectarse el nodo. Para ello, primero se lanza un mensaje que sigue un camino aleatorio, almacenando los nodos por los que pasa. Esos nodos forman la lista de candidatos. Después, se aplican unas métricas que determinan el atractivo de cada nodo según su grado y su carga. Finalmente, el nodo escoge sus nuevos vecinos al azar entre los candidatos, de tal forma que la probabilidad de ser escogido de cada nodo es proporcional a su atractivo.

A.3.5 Resultados experimentales de DANTE

Mediante simulación hemos comprobado el rendimiento de DANTE. En concreto, en nuestros experimentos hemos podido observar como DANTE efectivamente adapta su topología a las condiciones de carga. Las topologías formadas por DANTE son muy eficientes ya que concentran muchas conexiones en un número pequeño de nodos, lo que hace que las búsquedas puedan ser resueltas en pocos saltos. La razón es que los nodos bien conectados conocen mucho acerca de la red y pueden responder a muchas de las búsquedas que reciben. A su vez, DANTE también evita que los nodos bien conectados se vean saturados, primero haciendo que sean los pares más capaces los que reciban las conexiones. Si aún así las búsquedas generan una carga demasiado alta, el mecanismo de reconexión hace que los nodos bien conectados pierdan conexiones para evitar su saturación.

También hemos estudiado si bajo circunstancias desfavorables el rendimiento de DANTE podía disminuir de manear importante. Primero, se efectuaron 'ataques' sobre los nodos mejor conectados. Se comprobó que DANTE reaccionaba correctamente rehaciendo las conexiones para formar una nueva topología también eficiente. Después se ejecutaron experimentos en los que la frecuencia de entradas y salidas de nodos de la red era grande, una circunstancia que incide negativamente en otros sistemas P2P. Tampoco en este escenario el rendimiento de DANTE se vio muy afectado.

Finalmente hemos comparado DANTE con GIA [14], otra propuesta de sistema P2P adaptativo bien conocida por la comunidad académica. Nuestras simulaciones muestran que DANTE, debido a su tendencia a formar topologías centralizadas, da un rendimiento mejor que GIA, cuyo funcionamiento sigue heurísticas distintas.

A.4 Conclusiones

Tanto el modelo para caminos aleatorios como el sistema P2P adaptativo que se perseguían como objetivos de esta tesis han sido conseguidos.

El modelo toma una aproximación innovadora al basarse en una serie de métricas que caracterizan un camino aleatorio en cada salto del mismo. Nuestras simulaciones han probado que las estimaciones predichas por el modelo se ajustan bien a los resultados reales. Por otro lado, las mismas simulaciones han mostrado que el comportamiento de los caminos aleatorios no viene dado únicamente por la distribución de grado. El cómo se haya construido la red también influye en ese comportamiento, lo que se vio al comparar los resultados de redes de Barabasi y Newman.

Asimismo, el sistema de red auto-adaptativa propuesto en esta tesis, DANTE, consigue formar topologías eficientes para la resolución de búsquedas en una red P2P no estructurada. DANTE adapta la topología a la carga de la red, acortando la longitud media de las búsquedas concentrando muchas conexiones en unos pocos nodos, los más capaces de la red. A su vez, también previene que ninguno de dichos nodos sea saturado haciendo que pierdan conexiones si están cerca del punto de sobrecarga. Finalmente, también reacciona de manera adecuada a ataques dirigidos contra nodos de alto grado, y su eficiencia no se ve fuertemente afectada incluso en circunstancias poco favorables.

Appendix B

GNU Free Documentation License

Version 1.2, November 2002

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that

the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License

requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license

notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role

of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the

Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the

Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a

unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document

within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.