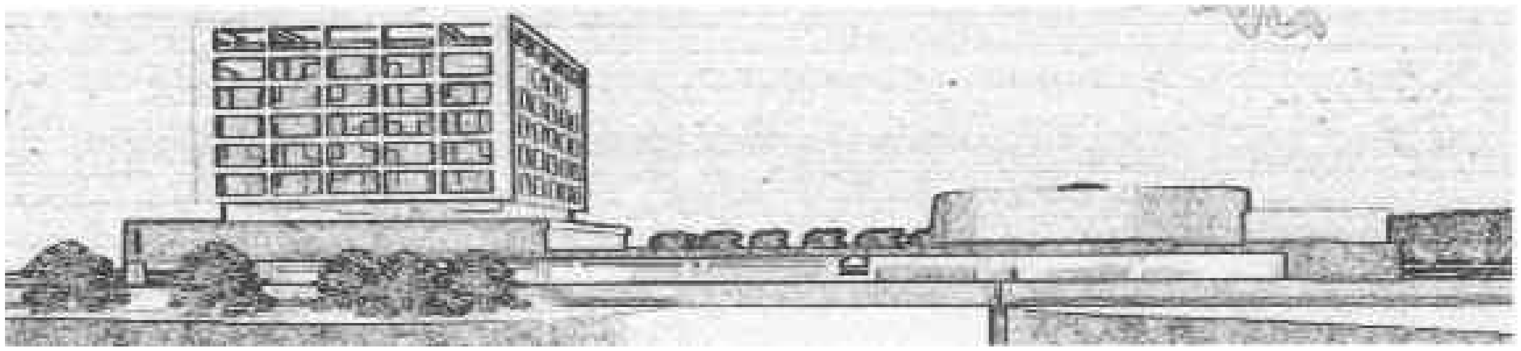
VOLUME VII, NUMBER 12



REPORTS ON SYSTEMS AND COMMUNICATIONS

DANTE: Self-Adaptable Unstructured Peer-to-Peer Networks

LUIS RODERO MERINO
LUIS LÓPEZ
ANTONIO FERNÁNDEZ
VICENT CHOLVI

Móstoles (Madrid), Spain, Jul. 2005
Depósito Legal: 50653-2004
ISSN: 1698-7489

DANTE: Self-Adaptable Unstructured Peer-to-Peer Networks

Luis Rodero Merino*, Luis López*, Antonio Fernández*, Vicent Cholvi†
*LADyR
Universidad Rey Juan Carlos
Mostoles, Spain, 28933
Email: {lrodero, anto, llopez}@gsyc.escet.urjc.es
†Universitat Jaume I
Castellón, Spain, 12071
Email: vcholvi@lsi.uji.es

Abstract— **This paper faces the problem of searching in unstructured peer-to-peer (P2P) networks. It introduces a novel paradigm of self-organized dynamic overlays whose topology adapts to the traffic on the network. The proposed mechanism has been proved to drive the system to an optimal topology when the network is in very high or in very low conditions. In this paper we present DANTE, a P2P system that uses this adaptation mechanism along with random walks for resource location. Simulation results show how DANTE achieves a better search performance than other P2P unstructured solutions. We have also developed a real implementation of DANTE. Using it we have checked the validity of our initial hypotheses, demonstrating that the expected behaviour is obtained in the executions of this implementation.**

I. Introduction

Peer-to-peer (P2P) systems have obtained great popularity during the last few years. The best known and most widely used application of P2P technology is the sharing of contents. This application was first explored by Napster [1], which grew extremely fast captivating thousands of users worldwide and raising an increasing interest in the scientific community. Other P2P solutions have appeared incorporating new ideas, like Gnutella [2], where, unlike Napster, no central server is present and search is done in a totally decentralized manner, or Kazaa [3], where several nodes act as superpeers, with different behavior and mission than the rest of participants.

The basic idea of P2P systems is that all peers are, at the same time, clients and servers, that is, they offer resources to other nodes and use resources from other nodes. P2P systems represent a new and powerful paradigm that differs from the traditional client-server architecture where each participant has a specific role. This paradigm has brought the necessity for novel solutions to deal with some of its limitations. More specifically, the research community has devoted many efforts to develop new and efficient techniques for the location of resources. Traditional location solutions, like using a centralized directory (Napster) or flooding (Gnutella), have shown serious drawbacks: vulnerability, lack of scalability, etc. Trying to solve those problems, researchers have proposed new searching mechanisms based on random walks or structured networks.

In this paper, we propose a search technique that combines search by random walks with a self-adapting topology mechanism. Networks where nodes change their connections, and hence the overlay topology, to adapt to changes on network conditions (such as traffic load) are said to have a *Dynamic Adaptable Network Topology*. In our solution, which we have called **DANTE**[1] the network topology evolves between a starlike and a random like one as the load varies. Note that previous results [4] prove that, assuming that each node knows its neighbors resources, the starlike topologies are optimal (in terms of search time) for non-congested systems and random-graph-like systems are optimal for high loads. For intermediate loads, hubs (nodes that have many more incoming connections than the average) appear. Hubs have a wide knowledge about the network contents, but are not central (not all nodes are connected to them), for this reason, they allow solving queries in a few hops without becoming as congested as central nodes.

The basis of the topology adaptation mechanism proposed here was introduced in [5], where an analytical evaluation is performed assuming that searchs follow shortest paths among nodes. Our proposal uses random walks in searchs, so peers do not need global knowledge about the network topology. Besides, in this paper we improve this mechanism proposing a more accurate congestion computation method that is suitable to be used in a real scenario. We also empirically demonstrate the validity of the theoretical hypotheses in [5] by developing a real working implementation of the system. Finally, we have created a simulation software that allowed us to compare the performance of our system with other unstructured P2P networks. Summing up, we show here the feasibility of our resource location proposal for real P2P systems.

This paper is organized as follows. In Section II we outline some solutions for resource search in P2P networks, describing the advantages of random walks on unstructured networks. In Section III we discuss *Dynamic Adaptable Network Topologies* and how they are combined with random walks to build an effective search solution. Other solutions with an approach similar to ours are also discussed. Section IV introduces

[1]From Dynamic Adaptable Network TopologiEs.

the reconnection mechanism that is the basis of DANTE. An analysis of how to predict load on nodes depending on network conditions in done in Section V. That analysis was used to design the simulations presented in Section VI, where DANTE is compared with other P2P solutions. Finally, the results obtained from the execution, under different setups, of a functional DANTE implementation are commented in Section VIII.

II. Location of resources in P2P networks

If we look into the mechanism used to locate resources, P2P systems may be classified into two categories:

- **Centralized**. A central repository stores an index of all resources available in the network, with the location of those resources (which nodes hold them). All nodes register their resources and address their searchs to this repository.
- **Distributed**. There is no central repository. Queries are forwarded through the network.

Clearly, distributed systems are closer to the idea of P2P. They are less vulnerable to attacks or censorship, and have better scalability, since the P2P system tasks are shared among all network participants. Decentralized systems may be classified again by the way new resources are located in the network:

- **Unstructured networks**. The placement of resources is not related with the network topology. Nodes are also added in a non regular way. Examples are Gnutella [2] and Kazaa [3].
- **Structured networks**. Resources are placed at precise locations. The location for every resource is computed by a hash function. Search is done by means of a distributed hash table (DHT). Examples are Chord [6], Pastry [7], Tapestry [8], CAN [9] and Kademlia [10].

A. Drawbacks of structured P2P networks

Several structured networks proposals have been developed by the research community. On these networks location of resources is deterministic, computed by some mechanism based on Dynamic Hash Tables. Resources are typically found in only $O(log(n))$ hops (being n the network size), and are always found if they are present in the network.

Despite their advantages, structured networks have failed so far to gain wide acceptance on P2P systems. This is because they present some drawbacks that prevent them to be deployed on typical content sharing systems. These drawbacks are enumerated (more or less exhaustively) in many P2P research papers like [11], [12] and [13]. Some of them are described here:

- **DHT tables are expensive to maintain**. When peers enter or leave the network DHT tables must be rebuilt, this implies some communication costs. If nodes move often from-into the system, the communication overhead becomes relevant.
- **Resource popularity is unequally distributed**. It has been observed that content popularity can vary strongly in content-sharing P2P networks [14]. The more popular some resource is, the more searchs 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 [15].
- **Resources are naturally replicated**. It can be expected than, the most 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 addressed always to the particular node the DHT table assigns the resource to.
- **Typical searchs 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 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 [16], but maintaining these indexes can be expensive in terms of communication and processing costs, as any term could 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 [13] 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.

Many of those drawbacks are faced by different works like the ones we cite above. But each one of these solutions seems to deal with a specific limitation of structured networks. It is unknown if these proposals can be combined, and if the resulting system would be still efficient.

Unstructured networks do not present any of these drawbacks, and the use of random walks can made them scalable providing at the same time a reasonable guarantee of finding resources under the proper conditions. Thus, we deem they can be a suitable solution for future (and present) P2P systems design.

B. Search in unstructured P2P networks

Unstructured networks are, nowadays, the most used for content sharing. They are simple to manage and seem to deal better with the organization overhead under high churn rate of peers. On the other hand, structured networks are more efficient locating resources and their search mechanisms scale much better than the traditional techniques used by unstructured networks like flooding or supernodes.

Three search methods are typically used on unstructured networks. The first search mechanism is flooding, used by Gnutella [2]. By wich each peer broadcasts its queries to all its neighbors. If some neighbor has the resource, it replies to the query source. If not, it forwards the query to all its neighbors again. Search is limited by a TTL mechanism. The main drawback of flooding is the lack of scalability (for a deeper discussion on this topic see [17]). The second technique uses the idea of *supernodes*: well-known nodes that store the index of all other peers on the system. When some node enters the system, it registers itself in a subset of all supernodes set. During the node's lifetime, all its searchs are addressed to supernodes. It is used by Kazaa [3]. The scalability of this system has yet to be analyzed, and is clearly more vulnerable to attacks that a pure P2P system. Finally, the third search mechanism is the use of random walks. In this case, nodes forward each query to only one peer, chosen randomly among its neighbors. These messages are typically called walkers. The requesting node sends $k \geq 1$ walkers. Each walker will follow its own path. This mechanism introduces less communication overhead compared with flooding, but it may also take longer to solve queries. In [18] and [19] we can find some comparisons of both techniques in different network topologies and conditions, concluding that random walks seem to be a promising technique suitable to solve the scalability problems of flooding in unstructured networks. The cost to pay is an increase on the search time for certain topologies.

III. Dynamic Adaptable Network Overlays

Random walks seem to be the most feasible search technique for unstructured networks in the future. But it is well known that the performance of this search technique is highly dependent on the overlay structure of the system [5], [20], [21]. The approach traditionally taken in the literature to model this begins by assuming that nodes know their own resources plus the ones held by their immediate neighbors. In this case, if some peer becomes a central node (all participants are connected to it) it will know all the resources present in the whole system and will be able to correctly answer all queries. In a starlike topology a few nodes become central and all nodes in the system are connected only to them. Hence, all searches are solved with at most one hop.

With this argument we understand that, in a non congested scenario, the optimal topology is a highly polarized starlike structure. However, this situation is inefficient if congestion considerations become relevant, since the central node may become overloaded. This is supported by the results in [19], where it is shown that high-degree nodes (those having most connections) support most of the shared load. Moreover, it has been proved in [4] that the optimal network topology is a homogeneous-isotropic one in the presence of severe congestion.

We introduce in this paper a P2P system which implements a reconnection technique that adapts the overlay network topology to the load on the network. The resulting topology tends to a starlike when congestion is small and to a random-like when congestion becomes relevant. For intermediate loads, some peers become hubs, that is, nodes that have much more incoming connections than the average degree on the P2P system. Hubs have a wide knowledge about the network contents, but are not central (not all nodes are connected to them), for this reason, they allow solving queries in few hops without becoming as congested as central nodes.

A. Related work

There are other proposals in the literature of P2P systems where network topology changes to adapt itself to the network load. In all of them random walks are used as the search mechanism. For example, Lv *et al.* [22] present a system where a flow control mechanism avoids nodes to become overloaded and changes the topology making messages to flow toward nodes with higher capacity. To achieve this, every node periodically tracks the messages it receives. If the node is overloaded, it redirects the most active neighbor (the one sending more queries) to another neighbor with higher spare capacity. In this way, high-capacity nodes tend to have higher degrees, and no node is overloaded. This solution, nonetheless, requires every node to know the state of all its neighbors, which introduces an important communication overhead. The simulation results presented in that work do not take into account this overhead. In any case, their results support the idea of improving P2P networks efficiency taking advantage of nodes heterogeneity as we do.

Chawathe *et al.* in [12] propose a system called **Gia** that strives to avoid overloading any of the nodes by explicitly accounting for their capacity constraints. Walkers are explicitly forwarded to high-capacity nodes, which should be more able to handle requests. An active flow control avoids overloading hot spots: one node can send messages to some neighbor only if it has notified the sender that it is willing to accept it. Topology is also adapted in a continuous manner. Nevertheless, their mechanisms introduce more network overhead than ours. In our system, when some node becomes overloaded, their neighbors disconnect by their own, so no explicit flow control is needed. Besides, in Gia nodes need to be aware of the state (number of connections) of their neighbors. The performance in a real scenario, where the added overhead should be considered, is unknown.

Both solutions are interesting and show the potential of using random walks on networks built by *dynamic network topologies*. Nevertheless, our proposal simplifies the reconnection mechanism, does not require nodes to report their state to their neighbors periodically, and tends to the optimal topology for all load conditions. We have implemented a simulator of the Gia system to compare its performance with the one of our system. More information about the simulators, experiments results and some comments can be found in Section VI. This comparison is specially interesting as Gia creators have already shown that their system performs better than flooding or searching using supernode mechanisms (as in Kazaa).

IV. DANTE Reconnection mechanism

DANTE, the P2P system we propose, is based on the creation of a dynamic adaptable overlay by means of a mechanism based on the one presented in [5]. Nodes have two kind of links: incoming and outgoing connections. A connection from node N_1 to node N_2, $N_1 \rightarrow N_2$, is said to be an outgoing connection for N_1 and an incoming connection for N_2. Messages can flow in any direction of the link. Nodes do not change any incoming connection once it has been accepted, they can only manage their outgoing connections. It is possible to have a $N_1 \rightarrow N_2$ connection and a $N_1 \leftarrow N_2$ connection open at the same time.

The number of outgoing connections of each node is fixed. Nodes manage those outgoing connections, changing them using a particular algorithm. The interconnection topology changes as nodes adapt periodically these outgoing connections. The algorithm used to choose which peers each node connects to is based on an *attachment kernel* Π_i, which determines the probability of a particular node to be connected/rewired to node N_i. The proposed kernel has the form

$$\Pi_i \propto k_i^{\gamma_i} \tag{1}$$

where k_i denotes the number of links of node N_i and

$$\gamma_i = \begin{cases} 2 \text{ if } N_i \text{ is not congested} \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

If some node receives more queries that it can process, we say that node is *collapsed* or *congested*. The algorithm tries to make well connected nodes more attractive, so outgoing connections will tend to point to those peers as they know more about the network. But if some node gets collapsed, then it stops being attractive, and neighbors will disconnect from it quickly.

The rationale behind Equations 1 and 2 is explained as follows. First, remark that we assume that each node knows its resources and the resources of its one-hop neighbors. We note that by taking a value of $\gamma_i = 0$ for all nodes, we obtain a random topology (intuitively, all nodes have the same probability of being chosen for a new connection [23]); in turn, if the value of γ_i is strictly greater than 1, we obtain a starlike (the more connections one node has, the more likely it will be chosen by other nodes, finally building a starlike topology [23]). Consequently, in [5] is established that the value of γ_i will be either 2 if the node is not collapsed and 0 otherwise. Thus, the network will evolve towards a random-like topology when the nodes become collapsed, or towards a starlike topology otherwise. It is easy to realize that the value of γ_i for not collapsed nodes has a strong impact on the way topology evolves.

A. Congestion computation

Using traditional queuing theory [24] we say that some node is *collapsed* when it recieves more requests than it can process.

The number of requests received by node N_i, λ_i is easy to obtain. On the other hand, the node processing capacity, can be more difficult to determine. There are two manners.

First option is to use a fixed *Threshold* as a node parameter. If the number of received queries in a recent period is greater than the node's threshold, then that node is collapsed. This threshold should be based on the node's capacity, or the capacity the node's owner wants to devote to the P2P system. We can use fixed thresholds to force the network to form starlike or random topologies setting the proper threshold on nodes: setting a small threshold will lead to a random topology as no node can have many connections (as soon as they get a few connections, and so a few queries, they would become collapsed very quickly); setting a large threshold will lead to a starlike topology (nodes are not collapsed although they receive many queries, so they keep being attractive for incoming connections). Let λ_i the number of queries received by node N_i in the last time unit, and $Threshold_i$ be that node's congestion threshold. Then we say that:

$$\text{if } \lambda_i > Threshold_i \Rightarrow N_i \text{ is congested} \tag{3}$$

The aproach described above was used in [5], but can be unaccurate in some circunstances. First, the threshold must correspond with the real node processing capacity. Second, the load of some node also depends on the knowledge the node has: the more the knowledge, the more time it will take to locally search for some resource, this is, to process some query. That knowledge depends on the number of connections: the more connected some node is, the more knowledge it has. Hence, well connected nodes will need more time to process the same number of queries than loosely connected nodes. Finally, the bandwith should also be considered, as forwarding queries or sending results can be a time consuming task.

We have improved the congestion computation method, in order to better suit real systems. This method uses an adaptable congestion computation mechanism. Each node keeps track of the time to process queries received during the last period. Let μ_i be the processing rate of node N_i, then we say that:

$$\text{if } \lambda_i \geq \rho\mu_i \Rightarrow N_i \text{ is congested} \tag{4}$$

where we denote $\rho \in [0, 1.0]$ as the *Congestion acceptance rate*. This parameter represents how much load a node will take before becoming a collapsed node. In queuing theory [24] ρ is denoted as the *occupation rate* or *service utilization*. In the next section we describe how μ_i should be computed.

B. Queries processing rate

To compute the queries processing rate, μ_i, as we said before, is not enough to keep track of the time consumed in searchs. Bandwidth must also be considered.

When some node receives a query (search message), it checks if it knows the resource the query is referred to. We call this checking the *lookup task*. If after this lookup the resource has been found then a success message is sent to the query origin peer. Otherwise, the search message TTL is decreased and checked. If the TTL is 0, a 'search failed' message is sent to the origin peer, if not, then the search message is forwarded to some randomly chosen neighbor. Whatever the case, a message must be sent through the network. Depending on the node's upload bandwidth, this task can need more time than the lookup task. Then, it may happen that the queries bottleneck is not the lookup task, but the message sending process. Let BW_i be N_i's bandwidth, and PS the packets max size.

Then, we define $MSPT_i$, the mean search processing time, as:

$$MSPT_i \,=\, max(\,PS/BW_i\,,\,MLT_i\,) \tag{5}$$

where MLT_i is the mean lookup duration in node N_i. It could be argued that $MSPT_i$ should be defined instead as $MSPT_i \,=\, PS/BW_i \,+\, MLT_i$. But in modern computers we can assume that search processing and packet sending run in a pipeline, so effective $MSPT_i$ is the one defined in Eq. 5.

Hence, we can define μ_i as:

$$\mu_i \,=\, 1\,/\,MSPT_i \tag{6}$$

Our simulation software computes μ_i as it has been described here.

C. Candidates search

The reconnection algorithm is applied over a set of candidates, that is a subset of all the peers in the network. The set of candidates can be built using different techniques. If walkers store the state of nodes they traverse, peers could use that information to keep a cache of known peers as walkers cross them. A Ping-Pong message mechanism like the one used in Gnutella could also be used, but this would introduce some communication overhead.

Finally, nodes could send special node search messages that do not look for resources, but for peers. Those messages would be sent before each reconnection is started. When the TTL of the search reaches 0, a message with the information of all nodes traversed is sent back to the node that originated the search. With a high probability the message would have traversed well connected nodes, that are the best candidates to connect to. Besides, the communication overhead would be small and constant. This is the solution we have chosen for our system. In Section VII-D we will see that the P2P system performance is not penalized when using it. In fact, we get some surprising results.

V. Analysis of nodes load for random and starlike topologies

In this section we try to cuantify some of the characteristics of P2P networks, mainly those referred to load on nodes. We will use the results obtained here to design the simulations in Section VII-A.

In the previous section we showed how μ_i should be computed. Now we will see how λ_i can be predicted depending on network conditions, this is, load and topology.

First, we compute how many queries each node will get depending on the network topology. Then, we will particularize our computations for the two extreme possible topologies: random and starlike.

A. Load and processing capacity of nodes

Let l be the mean search path length. Let f be the query generation rate for all nodes. Let n be the number of peers. Then, we say that

$$\text{Total Load on the Network} \,=\, nfl \tag{7}$$

Let δ_i be the degree of node N_i. Then:

$$\text{Total Links on the System} \,=\, \frac{\sum_{j=1}^{n}\delta_j}{2} \tag{8}$$

So, we can define the load per link as:

$$\text{Load per Link} \,=\, \frac{2nfl}{\sum_{j=1}^{n}\delta_j} \tag{9}$$

Note that load per link refers to the number of messages that traverse some link in both directions. So each node receives Load per Link/2 messages per neighbor. Then, the toal load for some node N_i is:

$$\lambda_i \,=\, \frac{nfl}{\sum_{j=1}^{n}\delta_j}\delta_i \,+\, f \tag{10}$$

where the first operand is the load due to searchs that traverse N_i, and the second operand refers to locally started searchs.

B. Random topology

In a random topology we can assume that nodes degrees are roughly the same for all nodes, $\forall i\ \delta_i \,\approx\, \delta$. Then, $\sum_{j=1}^{n}\delta_j \,=\, n\delta$. So, we can rewrite Eq. 10 as:

$$\lambda_i \,=\, \frac{nfl}{n\delta}\delta \,+\, f \tag{11}$$

simplifying, this becomes:

$$\lambda_i \,=\, (l+1)f \tag{12}$$

To compute l, we can assume it to be the expected mean of a geometric distribution. Let r be the resource replication factor, as defined in [12]. This is computed as $r = k/n$, where k is the number of nodes that have a particular resource (in our simulations and experiments we will assume that all resources have the same replication factor). Then, we can say that:

$$P_i \,=\, 1-(1-r)^{(\delta_i+1)} \tag{13}$$

where P_i is the probability of finding the resource on node N_i. However, there is something to note. The $(\delta_i\!+\!1)$ exponent emerges from the fact that each node will check its own list of resources plus the list of resources of its neighbors (remember that each node knows the resources of neighbour peers). Nonetheless, when some node N_i is processing a resource search message, it will neither check its own list of resources nor the sender's. The reason is effiency, the sender would not have forwarded the query in case N_i or the sender itself had the resource in its local resources list, so it is not needed to check those lists. Then, Eq. 13 only holds for locally started searchs. When processing search messages P_i is defined by:

$$P_i = 1 - (1-r)^{(\delta_i - 1)} \tag{14}$$

for simplicity reasons we will use Eq. 14. Thus, as $\forall i \; \delta_i \approx \delta$, we can say that $\forall i \; P_i \approx P$, and hence we can define l as:

$$l = \frac{1}{P} = \frac{1}{1-(1-r)^{\delta - 1}} \tag{15}$$

C. Starlike topology

For this topology, we focus on load on central nodes. First, we must be aware that central nodes do not forward any query. This is because all nodes are connected to them, so they have all the knowledge about resources present in the network, and hence they do not need to forward searchs. Let z be the number of central nodes. We can rewrite Eq. 7 as

$$\text{Total Load on the Network} = (n-z)fl \tag{16}$$

we should also realize that there is not search traffic through connections between central nodes. Besides, each peripheral node is connected to all central nodes and only to them, and search traffic trough these connections only moves in one direction. Thus, we rewrite Eq. 9 as:

$$\text{Load per Link} = \frac{(n-z)fl}{(n-z)z} = \frac{fl}{z} \tag{17}$$

finally, applying Eq. 10 we compute the load on central nodes. Let N_i be a central node. Its degree δ_i is $n-1$ as all nodes are connected to it. But it will only receive searchs from peripheral nodes, so we can assume its degree to be $n-z$. Finally, all searchs will be solved in one hop[2], so $l = 1$. Then, we can define N_i's load as:

$$\lambda_i = \frac{(n-z)fl}{(n-z)z}(n-z) + f = \frac{fn}{z} \tag{18}$$

VI. Simulations configuration

In this section we present some results obtained from the simulations of DANTE, Gnutella and Gia. A simulator was implemented for each system. All three simulators use two common small libraries we developed: one for the simulation of discrete events, and the other to implement concepts as message, link and node (with a sendMessage primitive). All the software was implemented in Java.

Simulations use the microsecond as the minimum unit of time. All nodes in all three simulators are configured with a given processing capacity, measured in operations performed by microsecond, and an upload bandwith, measured in bits by microsecond. Nodes performs tasks. A task can be the processing of an incoming message (a resource search message, a connection message, etc.) or a locally started process (a locally started search, a new reconnection process, etc.). When performing any task the node is said to be busy. Any other task started while the node is busy is enqueued until the running task is over. Sometimes tasks require to send some message to some other peer. That is also a time consuming task. When the node has finished the present task, it looks for any other pending task. Local tasks have priority over incoming messages. If there are not pending local tasks, then the messages queue is checked. Connection messages have priority over search messages. Messages queues are FIFO, except the search messages in Gia (see Section about VI-B Gia below).

Hence, there are two different times we take into account: processing time and packet sending time.

The processing time depends on the tasks being performed. Tasks that are not searchs for resources are assumed to last one unit of time. Searchs for resources, (that we called *lookup task* in Section IV-B), take a time proportional to the number of resources checked. Let C_i be the processing capacity of node N_i. Then, a lookup where m resources are checked would need $m \,/\, C_i$ units of time in node N_i. If the resource is not found, m will be the number of resources known by the node. If the resource is found, m will be exactly the number of resources compared until there was a match.

The packet sending time depends on the node's upload capacity and packet size. We set the message size to 1024 bits for all experiments.

To simulate network latency we make the following. Once a message is sent, it is not delivered inmediately by the 'network level', it is delayed for a time, the same for all messages, that can be configured for each experiment. This time is not related with the node bandwidth. We have set this time to be 300 milliseconds for all experiments.

Searchs are started by each node with a frequency set as a parameter of the experiments. When starting a new search, nodes first check if the resource is known locally. If not, a search message is built and forwarded to some neighbor.

Resources to be searched are chosen at random. Resources are distributed with a certain replication factor, that determines how many nodes hold some resource. For our simulations, all nodes hold the same number of resources. The replication factor is $1/n$, that is, each resource is hold by only one node.

All simulations last three hours of virtual time. Searchs started during the first 15 minutes of after the 75 minutes of virtual time are not taken into account for statistics.

A. Gnutella simulator

In Gnutella simulations, a network is built at the beginning of the simulation. Then, searchs are started by each node. A TTL based mechanism bounds the searchs scope. When some node receives a search message, it sends a reply to the search origin if it knows the resource. If not, it forwards the search to all its neighbors but the sender, unless the TTL has reached 0. Nodes keep a memory of the messages that already have

[2]More precisely, $l \leq 1$, as some queries started at no peripheral nodes can be solved locally, without creating a new search message. Thus, $l = 1-(1-r)^{z+1}$. But normally $n \gg z$ and so $l \approx 1$.

traversed them. If the same message arrives twice (or more times) to some node, it is silently discarded.

Note that in Gnutella the bandwith becomes much more important as a bound for query processing, as many more messages are forwarded than in Gia and DANTE systems.

Network topology does not change during the experiment after is set. For our experiments we use a random topology.

B. Gia simulator

Our Gia simulator implements the mechanisms described in [12]:

- A flow control mechanism, where each node periodically assigns flow-control tokens to its neighbors. Each token represents a query that the node is willing to accept. Tokens are distributed in proportion to neighbors capacities. Also, whenever some node creates a new connection, or disconnects from any neighbor, a new token reassignation is performed by that node. If some node receives a search from any neighbor that does not have tokens, that search is discarded.
- Biased random walks. Searchs are forwarded to those nodes with highest capacity. Besides, each node remembers to which neighbor it has forwarded each query. If the query arrives again to the node, it will be forwarded to a different neighbor.
- Topology adaptation protocol. The time between reconnections for each node depends on its *Satisfaction level*. Connections are made by a three-way handshake, where each node decides if it accepts the other as a neighbor. The number of neighbors is bounded by two params, *min_nbrs* and *max_nbrs*. When accepting a new neighbor, another connection may be drop.

All these mechanisms are implemented following the algorithms explained by Gia creators. But our implementation differs from what is described in [12] in some points:

- In [12] searchs are supposed to send keep-alives messages to the origin node as they traverse the network. This is done because of the assumption that peers can fail and so searchs can be lost. As we do not simulate nodes failures in our simulations, this messages are useless, so we decided not to penalize Gia introducing them.
- Nodes need to know their neighbors state periodically. We assume that this information can be included inside the messages for tokens assignations. Again, this frees Gia from unneeded communication overhead.
- Gia creators assume that 'client capacity is a quantity that represents the number of queries that the client can handle per second'. Instead of setting it as a fixed number in nodes configuration, we compute it dinamically for each node depending on its processing capacity, bandwidth and knowledge (the more knowledge some node has, the slower can process queries so the lesser capacity it is assigned to it).
- Gia paper stablishes that nodes keep a cache of known nodes that is updated by Ping-Pong messages. For our experiments with Gia we assume global knowledge, so all nodes are candidates in each reconnection. Ping-Pong messages are not used.
- Token allocation asignment is based on Start-time Fair Queuing [25] (SFQ), where each peer is assigned a weight equal to its capacity. In their simulator Gia creators use SFQ to assign tokens one by one to each node neighbors. In our simulator this would make Gia performance to sink due to communication overhead. Instead, we send tokens assignations periodically, and enqueue searchs in Gia nodes using SFQ. As before, each node is assigned a weight equal to its capacity. We think this is loyal to the Gia creators original idea.

We would like to add that, although Gia creators let us their simulator code, that has been very useful to clarify some concepts about Gia, our software differs deeply from theirs as it makes what we think more reallistic assumptions about processing and communication costs.

C. DANTE simulator

Our DANTE simulator implements the reconnection mechanism described in Section IV. Congestion is computed by a fixed threshold or dinamically, depending on the experiment configuration. Candidates for reconnections are determined by glogal knowledge or using nodes search messages. Again, this depends on the experiment configuration. Our system uses 'real' random walks, in the sense that we do not address searchs to highly connected nodes nor remember where searchs are forwarded to. Nodes only avoid, when possible, to forward the search message to the node that sent it.

VII. Simulations results

Here we will present and comment some results we have obtained with our simulations.

A. Flooding against random walks

First, we execute some experiments with relatively few nodes to compare the gnutella search results against GIA and DANTE random walks based searchs. Note that experiments with more nodes when using flooding would increase exponentially network traffic and so the mean search times. That is, experiments with more nodes would lead to worse results for Gnutella.

This set of experiments is run with 100 nodes. TTL is set to 100 for DANTE and GIA, to 4 for Gnutella. Each node in Gnutella stablishes 3 connections. For DANTE, each node manages 3 outgoing connections. In Gia the minimum number of neighbors will be set to 3, the maximum to 6, so Gia and DANTE have the same number of links.

For these experiments, all nodes have the same processing capacity, 1 operation per unit of time. That is, exactly one resource is checked per unit of time when processing some search. Nodes also have the same number of resources: 5000. All nodes have 5000 resources. Replication factor is set to 1, so each resource is located only in one node.

DANTE has been executed using fixed thresholds: 0 to force a random topology, 1000000 to force a starlike topology, and

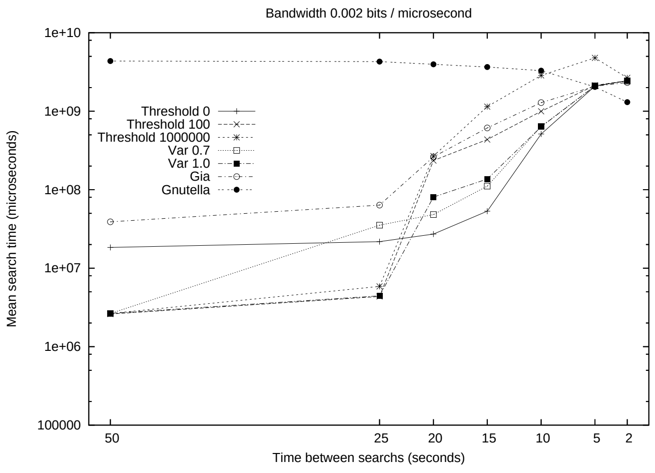


Fig. 1. Search times, bandwidth of 0.002 bits per microsecond

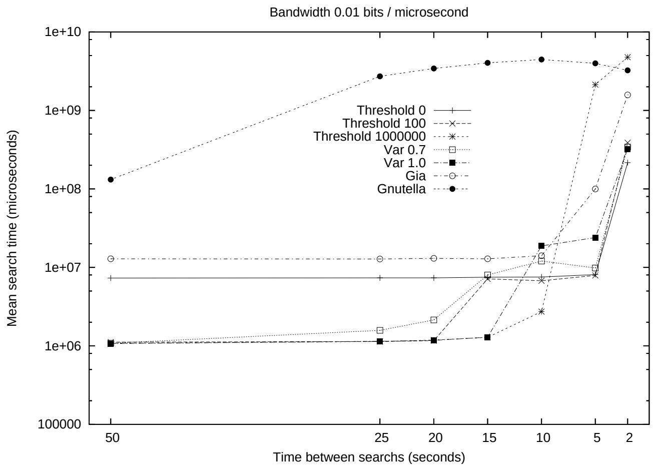


Fig. 2. Search times, bandwidth of 0.01 bits per microsecond

100 to make the topology dependent on the load. DANTE has also been executed with adaptable thresholds, using two different *Congestion Acceptance Rates* (ρ): 0.7 and 1.0.

Our goal was to compare the simulations results when the nodes performance is bounded by their processing capacity or their upload bandwidth (see Eq. 6), for random and starlike topologies. As the processing capacity is fixed, we compute the corresponding bandwidth BW for each case (all nodes will have the same bandwidth, $\forall i\ BW_i = BW$). These bandwidths have been computed using the Eqs. 4, 6, 12 and 18 defined in Sections IV-B and V. The resulting bandwidths are:

- $BW = 0.002$ bits/microsecond. This bandwidth makes the communication the bottleneck for random topologies.
- $BW = 0.01$ bits/microsecond. This bandwidth makes the queries processing the bottleneck for random topologies.
- $BW = 0.02$ bits/microsecond. This bandwidth makes the communication the bottleneck for starlike topologies.
- $BW = 0.1$ bits/microsecond. This bandwidth makes the queries processing the bottleneck for starlike topologies.

Finally, each experiment was executed with a different load, measured in units of time (microseconds) between resources searchs. The loads used are: one search per node every 50, 25, 20, 15, 10, 5, and 2 seconds.

The results are shown in Figs. 1, 2, 3 and 4.

For high loads, it seems that the mean search times decrease in some cases (see Gnutella in Fig. 1), or at least they remain constant. Of course, it is not so. Those results are due to the high congestion on the network, that makes that only brief searchs, those with low TTL, can finish within the frame of time we use for our statistics. This fact is confirmed by experiments logs.

By Figs. 1 and 2 we see that Gnutella gets worse (in orders of magnitude) mean search times. In Fig. 3 gnutella achieves a lesser mean time for the lower load than GIA and DANTE with 0 fixed threshold, both networks of random topology. The other networks, that have a starlike topology for that load, still perform better. For the next load Gnutella searchs get slower

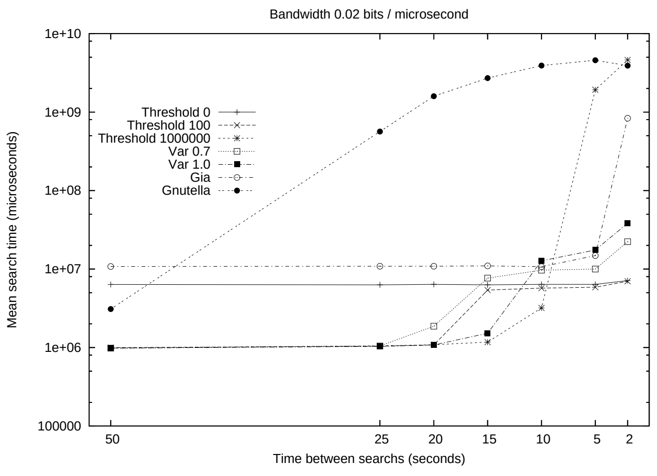


Fig. 3. Search times, bandwidth of 0.02 bits per microsecond

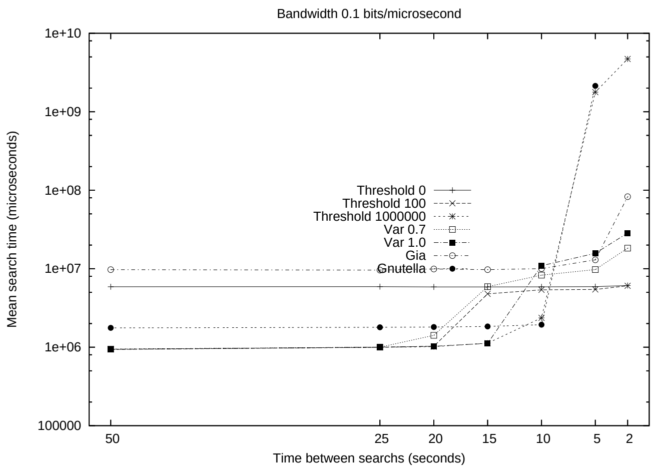


Fig. 4. Search times, bandwidth of 0.1 bits per microsecond

again.

Finally we analyze Fig. 4. First we should realize the configuration of this experiment sets the best possible conditions for Gnutella: few nodes and a high bandwidth totally devoted to resource searchs. Here gnutella performs relatively well for low loads, and yet worse that centralized networks: DANTE networks with big threshold or adaptable congestion computation with congestion acceptance rate of 1.

Then, it seems clear that random walks outperform severely the Gnutella network, even when conditions are optimal for the flooding search technique. In real world networks, with thousands of nodes and the available bandwidth shared with other kinds of traffic on the net, Gnutella would get much worse results. Practically all finished searchs were successful in all experiments for DANTE, Gia and Gnutella, so Gnutella has not provided any advantage in success rate either.

The comparison between Gia, DANTE with fixed thresholds and DANTE with adaptable thresholds is developed in the next sections.

B. DANTE and Gia

There are two main differences between DANTE and Gia:

- Gia uses an explicit flow control mechanism to avoid overloading peers. DANTE simply avoids (and changes) connections to congested peers. It could be argued that DANTE does not avoid nodes to get collapsed, only reacts to that situation. Nonetheless, note that the definitions of congestion given in Section IV-A allow to configure a limit on congestion that prevent nodes to devote all its capacity to searchs processing. This can be done by setting the appropiate fixed threshold or *Congestion Acceptante Rate*.
- Gia nodes try to set as many connections as possible, until the max_ngbrs bound is reached, and with peers with the highest capacity. Then, Gia advocates could reason that setting a higher bound would improve performance. But then, it would be enough in DANTE to also increase the number of outgoing connections for each node to $max\_ngbrs/2$. Thus, the mean degree of both networks would be the same for high query loads, where DANTE tends to form a random topology, so Gia topology would not be an advantage. Also note that for starlike topologies the number of central nodes is exactly the number of outgoing connections for each node, so for lower query rates DANTE would get better results as there would be more hubs or central nodes to share the load. Hence, increasing the number of connections can be a higher help for DANTE than for Gia.

Figs. 1, 2, 3 and 4 can be used to compare Gia and DANTE results. As expected, with low load (where central or well-connected nodes appear in DANTE), our system performs much better than Gia. As load increases, the difference becomes smaller but DANTE with adaptable thresholds performs generally as well, at least, as Gia. We think that the continous DANTE topology adaption, and the overhead due to tokens assignation in Gia justify this differences. Another advantage of DANTE is its simplicity compared with Gia. For example, it does not use explicit flow control, nor require nodes to know their neighbors state.

TABLE I
CAPACITIES AND UPLOAD BANDWIDTHS DISTRIBUTION FOR SIMS. WITH 1000 NODES

Capacity level	Percentage of nodes
1x	20%
10x	45%
100x	30%
1000x	4.9%
10000x	0.1%

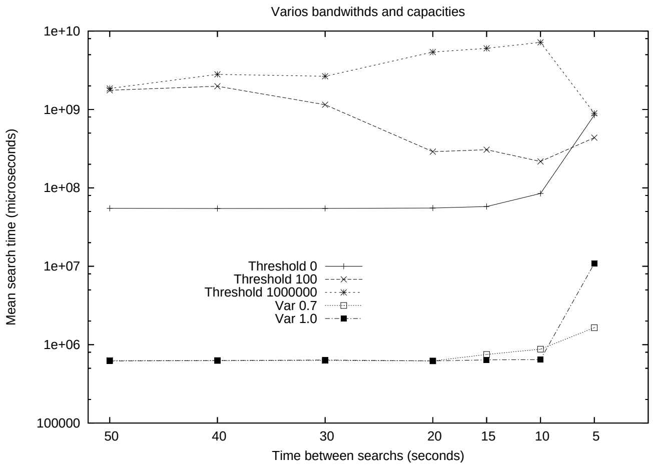


Fig. 5. Search times for DANTE, fixed and adaptable thresholds

C. Fixed and adaptable thresholds in DANTE

As it is said before, one of the main contributions of these paper is that not only shows the feasibility of the reconnection mechanism introduced in [5], but it also depures and improves it by adding an adaptable congestion computation technique more suitable to real networks. By results of first simulations it seems apparent that search performance is quite similar when using this congestion computation method and when using a fixed threshold of 100. This is mainly becacuse all nodes have the same processing capacity and bandwidth. Here some simulations are analyzed to show how the new method suits much better in networks where nodes capacities and bandwidths vary.

These simulations were run with 1000 nodes. These nodes were configured with different capacities and bandwidths, following a distribution similar to the one used in [12], that derives from measured bandwidth distributions of Gnutella nodes reported in [26]. The distribution is shown in Table I. Capacity level 1x means a processing capacity of 0.1 operations per unit of time and an upload bandwidth of 0.1 bits.

In Fig. 5 we can observe that simulations for fixed thresholds behave in a very rare way. Also, their performance has

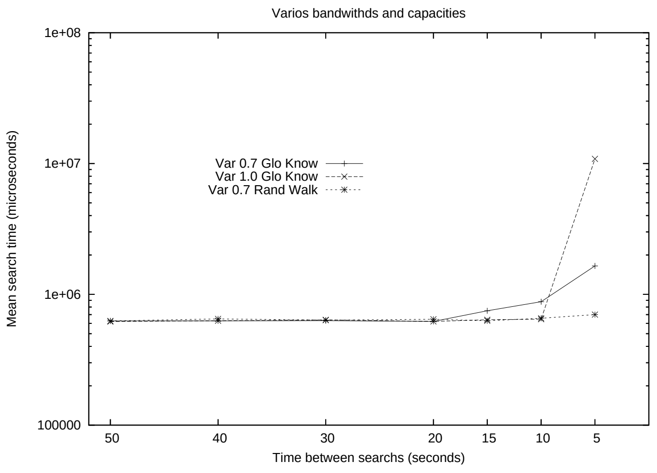


Fig. 6. Search times for DANTE with and without global knowledge

become extremely poor. The reason is that using a fixed threshold for node congestion computation makes the algorithm to assign connections regardless of the real capacities of nodes. Then, nodes with poor capacities can become well connected nodes although other peers, with more capacity, would be more able to handle incoming requests. As there are many more low capacity nodes than high capacity nodes in the system, we can be sure this will happen continually.

D. Node search cost

As it is explained in Section IV-C, DANTE nodes use random walks to locate candidates each time they wish to change their outgoing connections. When the node search TTL reaches 0, a message is sent to the origin node, with the list of all nodes traversed. Then the node applies the algorithm described in Section IV to choose to which nodes it must connect to, using the nodes in the list as candidates.

The previous simulations assumed some kind of global knowledge: all nodes knew all other nodes congestion state and so no search for candidates was performed. But in real conditions global knowledge is seldom available, so we need to analyze if searchs for cancidates would penalize DANTE performance.

We have run some simulations with DANTE where nodes do not have global knowledge and so they must perform candidates searchs. Those searchs were performed using random walks with a TTL of 100. These simulations were run with 1000 nodes, with the same capacities and bandwidth distribution described in previous section. The results are shown in Fig. 6.

In Fig. 6 we compare DANTE search times when using adaptable thresholds with global knowledge and ρ 0.7, 1.0 and DANTE using adaptable thresholds without global kwnoledge (and so performing random walks to look for candidate peers) and ρ 0.7.

The results are somewhat surprising: DANTE performs better without global knowledge. Nonetheless, the explanation is simple. First, candidate lookups add little overhead to the system. Second, when using random walks, the list of candidates is bounded by the node search messages TTL, in this case 100. Moreover, it is more likely that well connected nodes appear in that list more often (and possibly many times) that other peers [27]. Then, well connected nodes have to compete with few loosely connected nodes, and so they will be more easily chosen by the algorithm depicted in Section IV. On the other hand, when using global knowledge, well connected nodes must compete with all other nodes in the system, so there is still some chance that nodes with low degree are chosen. Thus, topology evolves slower to an optimal state and so the system performance is penalized.

VIII. Implementation execution results

Along with the DANTE simulator we have developed, also in Java, a functional system able to work as a real P2P system that implements the DANTE reconnection mechanism. Both the simulator and this real implementation are built following the same design, with the same main components. We have used this implementation to validate the results obtained with the simulator and also those predicted in [5].

In this section we present some results of executing this system with different fixed thresholds and loads. We have used fixed thresholds because our main goal was to compare the performance of different topologies under increasing loads. As we mention in Section IV-A, fixed thresholds can force the network to form a certain topology; very high thresholds are used for starlike topologies and null thresholds are used for random topologies. Intermediate thresholds make the topology evolve as the load changes.

The experiments were executed in a cluster of seven PCs, with 6 DANTE nodes running in each PC. Each node has three outgoing connections, and reconnections are performed every 30 seconds. In Fig. 10 the mean searchs times of these experiments are shown.

First, we verify that as we vary the load on the network the resulting topology evolves to adapt itself. From our executions we have captured three topologies, shown in Figs. 7, 8 and 9. The three topologies are obtained with the same fixed threshold. We see that the network in Fig. 7, which corresponds to a low load of 5 queries per node and minute, has a starlike topology. Fig. 8 shows the same network with a middle load of 10 queries per node and minute. With this load, central nodes get congested so the starlike topology moves to an intermediate topology with hubs. Finally, Fig. 9 shows the network under high load conditions, of 15 queries per node and minute. The network has evolved to a random topology where no node has many connections.

Now we analyze the mean search times obtained for different loads and thresholds. The data shown in Fig. 10 seem to be in accordance with what we would expect. For low loads, starlike topologies (those with a high threshold) have better performance than random topologies. Nonetheless, as the load increases, random networks outperform all other topologies. Clearly, this is because central nodes or hubs have become overloaded.

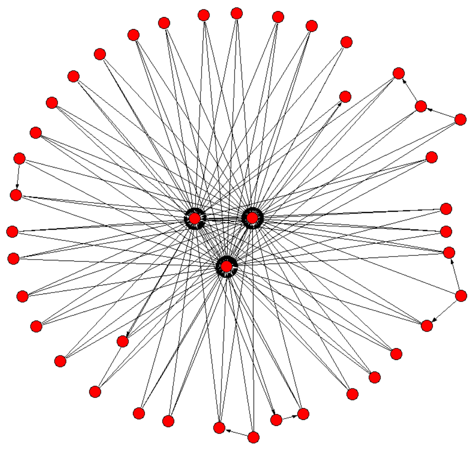


Fig. 7. Topology with low load, starlike

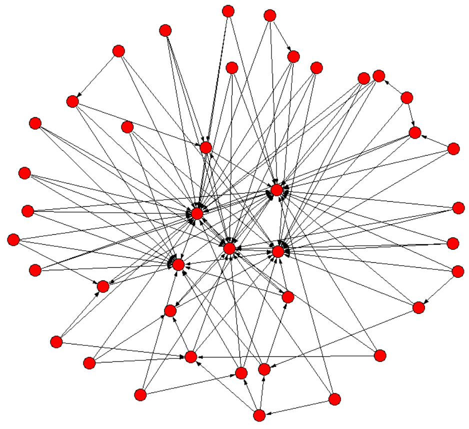


Fig. 8. Topology with medium load

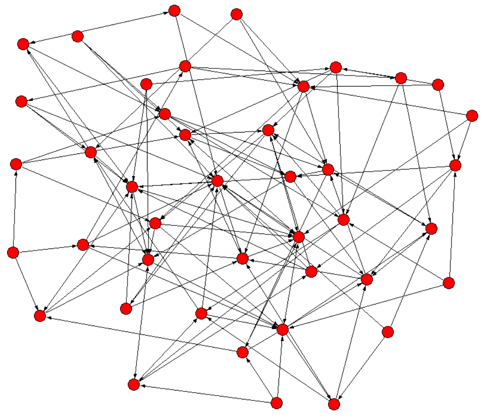


Fig. 9. Topology with high load, random

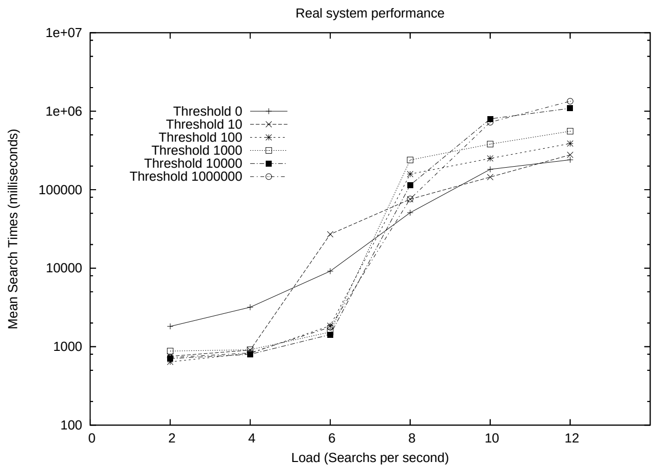


Fig. 10. Real system execution results

It is worth to note the results for threshold 10. It has a good performance, close to the best, for all loads except for load 6. That result was unexpected from the results in [5]. The simulations have shown a similar behavior: some thresholds achieve low search times except for certain middle loads, with which they obtain quite bad results.

After observing node logs, we think the explanation of this results are the continous sharp topology changes that happen for those certain loads and thresholds. Our reconnection technique tries always to form starlike topologies when possible. As some nodes get more connections, they become more attractive for other peers that also try to connect to them. When those nodes become overloaded, neighbors quickly disconnect. For certain loads and thresholds, this can happen too often: nodes become hubs very fast and so they receive many queries that must be enqueued, but if their threshold is too low they quickly get congested and then their neighbors disconnect. The high search times appear because the node, when neighbors disconnect from it, has already many search messages on its queue, messages that it will be unable to answer as it has lost almost all the knowledge it had. So, those messages will have to traverse the node queue and then be forwarded, probably to another hub that will also lose its connections in little time, and so on.

A possible solution we will try next will be to maintain the knowledge at a node even if its connections have changed. This could be simply done with a cache that maintains the freshest information available. A second technique, which could be combined with the previous one, is to prevent massive desconnections. For instante, the system could only allow one disconnection at a time.

IX. Conclusions and Future Work

We have introduced a new P2P system that combines random walks with dinamyc topologies to improve resource searchs efficiency. Our reconnection mechanism makes the network to form an optimal topology depending on query load conditions. This work successfully improves and evolves our original idea introduced in [5]. It also compares our search solution with other techniques and shows that it can be applied to a real system that performs as it is expected.

Our experiments show how our solution outperforms the flooding search technique by orders of magnitude. We have also compared DANTE with Gia, that uses random walks along with queries flow control and a different topology adaptation mechanism. Our system seems to perform better than Gia under almost all load conditions. Nonetheless, there are some ideas in Gia that we would like to adapt to DANTE, as the *Satisfaction* concept.

Further improvements to work on are: resource caches on nodes, to keep the lists of resources of previous neighbors; Bloom filters to compact the list of resources of neighbor nodes; search redirection policies that could perform better than random forwarding; search enqueuing policies other than FIFO and better node search methods (like agents).

Acknowledgments

The authors would like to thank Gia creators Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham and Scott Shenker for providing us with their simulator source code.

References

[1] "The napster website," http://www.napster.com. [Online]. Available: http://www.napster.com

[2] "The gnutella website," http://www.gnutella.com. [Online]. Available: http://www.gnutella.com

[3] "The kazaa website," http://www.kazaa.com. [Online]. Available: http://www.kazaa.com

[4] R. Guimera, A. Diaz-Guilera, F. Vega-Redondo, A. Cabrales, and A. Arenas, "Optimal network topologies for local search with congestion," *Physical Review Letters*, vol. 89, November 2002.

[5] V. Cholvi, V. Laderas, L. López, and A. Fernández, "Self-adapting network topologies in congested scenarios," *Physical Review E*, vol. 71, no. 3, 2005.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "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.

[7] A. I. T. Rowstron and P. Druschel, "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.

[8] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California, Berkeley, Tech. Rep., 2001.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "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.

[10] P. Maymounkov and D. Mazieres, "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.

[11] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, December 2004.

[12] Y. Chawathe, S. Ratnasamy, N. Lanham, and S. Shenker, "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.

[13] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "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.

[14] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. G. nad Henry M. Levy, and J. Zahorjan, "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.

[15] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "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)*, vol. 4, 2004, pp. 2253–2262.

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

[17] J. Ritter, "Why gnutella can't scale. no, really," Tech. Rep., electronic format in http://www.darkridge.com/ jpr5/doc/gnutella.html.

[18] G. H. L. Fletcher, H. A. Sheth, and K. Borner, "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.

[19] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "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.

[20] L. A. Adamic, B. A. Huberman, R. M. Lukose, and A. R. Puniyani, "Search in power law networks," *Physical Review E*, vol. 64, pp. 46 135–46 143, October 2001.

[21] C. Gkantsidis, M. Mihail, and A. Saberi, "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*, vol. 1, Hong Kong, March 2004, pp. 120–130.

[22] Q. Lv, S. Ratnasamy, and S. Shenker, "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.

[23] P. L. Krapivsky, S. Redner, and F. Leyvraz, "Connectivity of growing random networks," *Physical Review Letters*, vol. 85, pp. 4629–4632, November 2000.

[24] I. Adan and J. Resing, *Queueing Theory*. Available on-line, http://www.cs.duke.edu/ fishhai/misc/queue.pdf, 2001.

[25] P. Goyal, H. M. Vin, and H. Cheng, "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.

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

[27] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Physical Review E 64 026118*, 2001.