ARTICLE IN PRESS

Computer Networks xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Self-managed topologies in P2P networks ☆,☆☆

Luis Rodero-Merino [a,*], Antonio Fernández Anta [b], Luis López [b], Vicent Cholvi [c]

[a] Telefónica I+D, Tecnologias Emergentes Gestion Redes y Servicios, C/Emilio Vargas 6, 28043 Madrid, Spain
[b] LADyR, GSyC, Universidad Rey Juan Carlos, Móstoles, Spain
[c] Universitat Jaume I, Castellón, Spain

A R T I C L E I N F O

Available online xxxx

Keywords:
Self-adaptation
Self-management
Autonomic communication
P2P
Resource location

A B S T R A C T

The problem of efficient resource location is an important open issue in P2P systems. This paper introduces DANTE, a self-adapting P2P system that changes its peer links to form topologies where resources are located in an efficient manner via random walks. Additionally, this same self-adaptation capacity makes DANTE capable of reacting to events like changes in the system load or attacks on well-connected nodes by adjusting the topology to the new scenario. This adaptive behavior emerges as the global result of the individual work of nodes, without the intervention of any central entity or the need for global knowledge. Simulations show that this adaptation process makes the system scalable, resilient to attacks, and tolerant to a high transitivity of peers. Simulations are also used to compare this solution with other well-known self-adapting P2P system. From these results it can be concluded that the topologies achieved by DANTE offer better performance.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Peer-to-Peer (P2P) systems are bringing about a revolution in the way people develop and use network applications. The original client–server approach, where participants (also called *nodes* or *peers*) have a well-defined role as consumers or providers of *resources*, is being replaced by P2P architectures where each member can work simultaneously as both a client *and* a server.

P2P systems have interesting properties. First, they are inherently scalable, since all peers are susceptible of working as service providers. Second, they can be designed to work without any central control, which makes them more resilient to certain forms of attacks or node loss, as they do not present single points of failure. On the other hand, P2P systems also present challenging problems which cannot easily be solved with the current techniques. The most important of these difficulties is probably the so-called *search problem*, that is, the problem of efficiently locating a given resource or service in a P2P network, which may have an arbitrary structure.

Several solutions have been proposed to approach the problem of resource location. In them, peers are connected by virtual links forming an *overlay network* with a given topology. In this context, a particular location mechanism dictates how search messages are routed through the overlay network. Depending on the nature of the location mechanism, P2P systems have traditionally been divided into two main families [1,2]:

- *Structured Systems* [3,4,6]. These define an *identifier space* where each resource has a particular and well-defined *id*. The identifier space is partitioned in such a way that each node is assigned an *id* subset. Each node must know where all resources falling within its corresponding subset are located. Finally, the topology of the overlay network is carefully chosen so that search messages can be directed to find resources in a few hops.

☆ A preliminary version of this paper was presented at the ISPA 2006 conference.

☆☆ This research was supported in part by Comunidad de Madrid Grant S-0505/TIC/0285; Spanish MEC Grants TIN2005-09198-C02-01, TIN2008-06735-C02-01, TSI2006-07799 and TIN2008-03687; Bancaixa Grant P1-1B2007-44.

* Corresponding author. Tel.: +34 913373928; fax: +34 913374272.
E-mail addresses: rodero@tid.es (L. Rodero-Merino), anto@gsyc.es (A.F. Anta), llopez@gsyc.es (L. López), vcholvi@uji.es (V. Cholvi).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.09.006

- *Unstructured Systems*. In this case, there is no strict control over where each resource is located. Hence, non-informed search mechanisms are required.

Structured systems are usually very efficient given that they can locate resources in a small number of hops. Moreover, they do not produce *false negatives* (false negatives are associated with searches that fail to locate a given resource despite its presence in the P2P system). However, researchers have pointed out some limitations of structured systems [1,7]. For example, when there is a high churn of peers, the communication overhead can have a drastic effect on the network's performance. Furthermore, structured systems do not support keyword searches or complex queries as easily as unstructured systems. These drawbacks seem to make structured systems unsuitable for certain real-world scenarios like massive file sharing.

Unstructured systems are better suited to situations where there is a high rate of peers entering and leaving the network. However, they require different solutions to the problem of resource location. Two of the most popular solutions are the use of *flooding* and the use of *superpeers*. Flooding, used in the first versions of Gnutella [8], raises concerns about its scalability [9]. On the other hand, overlay networks based on superpeers have a very rigid structure, where all the peers are connected to a small subset of nodes.

Novel mechanisms have recently been proposed to overcome the limitations displayed by these solutions when it comes to solving the search problem. One of them consists in using *random walks* to route search messages: at each hop of the search message in the overlay network, the next node of the walk is chosen uniformly at random from among the current node's neighbors. The same process is repeated until the search reaches a peer that knows the location of the desired resource. Seemingly, random walks offer few guarantees for the search process, since it is usually not possible to know in advance how many hops are needed to find a resource. In addition, when a search fails (after a given time has elapsed or a predefined number of hops has been reached without finding the resource), there is no way of knowing whether the resource is not really on the network or if the result is a false negative. However, despite these limitations, previous work [10,11] has shown that random walks are a promising technique to solve the search problem in unstructured P2P systems. In this regard, it is important to notice that the overlay topology has a strong influence on the efficiency of random walks [12–14]. In consequence, some authors have proposed combining random walks with the use of *dynamic topologies* [14,7,15] (*i.e.* topologies that adapt to the load conditions by trying to optimize the performance of random walks). All these papers assume that each node is aware of its neighbors' resources. Observe that this is not a strong condition as it can be easily implemented in real networks.

For instance, Lv et al. [7] introduced a P2P system where nodes avoid congestion by means of a flow control mechanism that changes the topology, making messages traverse nodes with higher capacities. To do so, every node periodically checks its load. When a node is overloaded, it redirects its most active neighbor (the one sending most queries) to some of its neighbors with spare capacity. Thus, higher capacity nodes tend to have more connections and hence manage more queries.

Similarly, in **Gia** [15] (which is an evolution of the previous proposal), queries are forwarded to high-capacity nodes, which should be more capable of handling them. An active flow control mechanism avoids overloading hot spots: each node notifies its neighbors of the number of queries they are allowed to send, which depends on the node's spare capacity. The topology is adapted by a mechanism based on the nodes' *level of satisfaction*, which measures the distance between a node's capacity and the sum of its neighbors' capacities, normalized by their degrees.[1] This parameter determines whether or not each node will adapt the topology, and the frequency of these adaptations.

Finally, Cooper [5] proposes a self-adapting system where nodes have a degree proportional to the square-root of the popularity of the resources held by it (the popularity of a resource is the rate of queries for that resource over the total amount of queries in the system). This goal seems inspired by[11,20] where it is found that the resources replication strategy (amount of copies of each resource placed in the network) that achieves a minimum average search length is the one that, for each resource, places a number of copies proportional to the square-root of the resource's popularity. Cooper finds that those topologies perform optimally, in the sense that resources are found in the minimum average number of hops. We should note that Cooper does not assume that each node knows its neighbors' resources, which makes his work difficult to compare with other solutions like [14,7,15] or DANTE. Also, Cooper's work does not model nodes as entities with limited, and possible heterogeneous, processing capacities and bandwidths. Hence, for example, it does not take into account that nodes can become overloaded, neither does it avoid situations where nodes with low capacity get a higher degree (and so receive more searches) than other high-capacity nodes, which are more able to handle that load.

Our Approach. In this paper we introduce *DANTE*,[2] a proposal for an unstructured P2P system with self-adapting capabilities.

DANTE is inspired by the work by Guimerà et al. [13], which shows that, to obtain the best performance from a random walk, a P2P overlay topology must be either centralized (if the system is not overloaded) or random (otherwise). Based on this idea, DANTE implements a *reconnection mechanism* whose goal is to build the most appropriate topology dynamically by adapting it to the current network conditions. Hence, unlike previous systems, DANTE is capable of modifying the overlay topology from a centralized to a completely random one, depending on the network load. At the same time, DANTE avoids congestion at the network nodes without the need for any additional flow control mechanisms. This adaptation is performed in a totally decentralized manner. Note that even in the case where the topology that is built is central-

[1] A node's *degree* is the number of neighbors that node has.
[2] From Dynamic self-Adapting Network TopologiEs.

ized, the system remains a pure P2P network where all nodes have the same function (although some of them are better connected). Due to this, and as our simulations show, DANTE has the properties P2P systems are expected to have, such as the capacity to handle a high churn of peers and resilience to attacks.

This paper is organized as follows: First, in Section 2, we present DANTE and the rationale behind it. Secondly, by means of simulations, in Section 3 we show how the dynamic topologies built by DANTE provide significant benefits in terms of scalability, robustness and churn resilience. Finally, in Section 4 we draw appropriate conclusions.

2. DANTE, a new self-adapting P2P system

2.1. The rationale behind DANTE

In the work by Guimerà et al. [13], following an approach similar to the one proposed by Adamic et al. [12], it is assumed that nodes own named resources. It is also assumed that a particular node knows how to locate all its own resources plus the resources of all its neighbors (all nodes connected to the given one in the overlay). Hence, the search process is performed by a walker message which is routed among the overlay nodes following one of a set of predefined schemes (random, shortest path, etc.). A particular resource is found by the walker when the message arrives at a node that knows the location of the resource, that is, when the walker visits the node that owns the desired resource or one of its neighbors (the same assumption is made in other works that apply dynamic topologies, such as [7,15]). In this context, Guimerà et al. characterize the topologies that minimize the average search time when using a routing scheme based on random walks. They find that when the system is not congested (*i.e.* no node receives more searches than it can process), *the most efficient topology is a star-like structure*, where a small number of *central peers* are connected to the rest of the network. The reason for this is intuitive: if the topology is centralized, all searches are solved in just one hop of the walker, which is not delayed by congestion. In the same way they show that, when all nodes have the same capacity and under high load conditions, *the most efficient topologies are random-like*, where all nodes have a similar number of overlay links and the traffic is uniformly distributed. Additionally, they show that the transition of efficient topologies from centralized to random-like for increasing loads is sharp. Hence, they characterize the most efficient topologies in terms of load. On the other hand, they do not provide a mechanism for building such topologies in a real network. Note that this is not a trivial problem in P2P systems, where there is no global knowledge available and there is no central coordinator to tell each node which other peers it must connect to.

In a later work, Cholvi et al. [14] propose a reconnection mechanism that allows P2P systems to self-adapt to load conditions by converging to the topologies pointed out by Guimerà. By this mechanism, each node periodically decides which other peers it must connect to. This choice is based on the 'attractiveness' of each peer. Roughly, the attractiveness of a peer depends on its degree (the higher the degree, the more attractive the peer is, as it knows more about the resources in the network), provided that the peer is not congested. If, however, the peer is overloaded, it is assigned the smallest possible attractiveness value.

By simulation, Cholvi et al. show that their adaptation mechanism builds the topologies that were initially intended. However, their proposal has some important restrictions:

- They assume that each node has a global knowledge about the state (degree and congestion) of all the other members of the network. Clearly this is not true in many real networks.
- In their simulations the authors assumed that searches follow shortest paths, not random walks. Again, this cannot be fulfilled in real P2P systems, as the shortest path to the search destination is not known.
- Also, the networks used by Cholvi et al. for their experiments were homogeneous, in the sense that all nodes were identical. However, real world networks are known to be heterogeneous, with nodes with different processing capacities and bandwidths.

In [16] we presented the experimental results of a real implementation of a P2P system based on Cholvi's reconnection mechanism. This implementation differed in some respects from Cholvi's proposal, which made it closer to real scenarios. First, it used random walks to route search messages. Second, no global knowledge was available. Instead, the peers to connect to were chosen from a set of candidates C. This set was built at each reconnection by a special message that traversed the network following a random walk. When the message *Time-to-Live* (TTL, measured in the number of hops) expired, the list of visited nodes was sent back to the source node and became the set C for the reconnection.

Fig. 1 shows the topologies obtained by this implementation under different loads. As intended by the adaptation mechanism, with low loads the topology evolved to a centralized form, while under high loads the topology shifted to a random-like one. Furthermore, with certain intermediate loads the topology had a 'clustered' form with some well-connected, yet not central, nodes. The experiments also confirmed that, as predicted by Guimerà, those topologies were indeed the ones that allowed the lowest average search times to be obtained under the corresponding loads.

Nonetheless, we also became aware of some limitations of the reconnection mechanism proposed in [16]:

- A peer is considered to be congested if the number of searches received surpasses a certain fixed *threshold*. However, it is difficult to set such a threshold with accuracy a priori.
- Whenever the *threshold* is reached, the attractiveness of the node sinks to the minimum value. Due to this, all its neighbors tend to disconnect from it. This leads to sharp changes in the topology that have an impact on the network performance. For certain loads, this effect even prevents the topology from reaching a stable state.

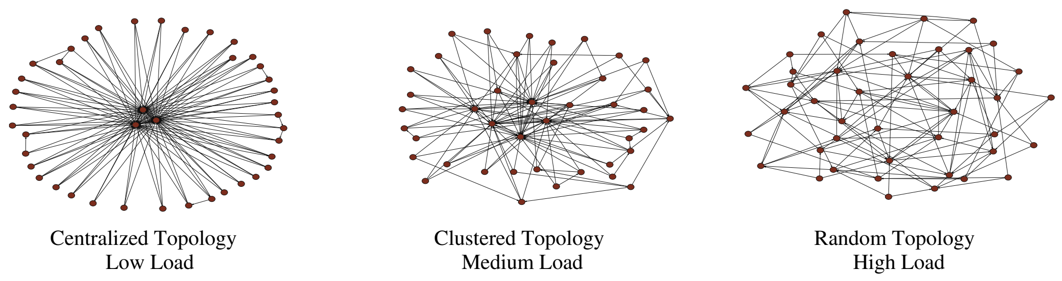

Fig. 1. Network topology adaptation.

Additionally, the experiments carried out in [16] were run with few nodes. Simulations with much bigger networks were necessary. We also realized that the system had to be tested under unfavorable conditions, such as attacks or high peer churns. Finally, it was necessary to test the reconnection mechanism in heterogeneous environments.

To address these issues, a new version of the DANTE P2P system was created with a different *reconnection mechanism* and then tested by extensive simulation.

2.2. DANTE's reconnection mechanism

This section introduces DANTE's reconnection mechanism, which dictates how each node must manage its connections.

Each node in DANTE handles a set of connections, which are denoted as the *native* node's connections. When a node *a* points one of its native connections to another node *b*, then that connection becomes one of *b*'s *foreign* connections. Thus a node can have both native and foreign connections, each linking it with some other peer in the network. Each connection is always native at one of its edges and foreign at the other. This differentiation is only meaningful for the reconnection process. Nodes can only change their native connections, not their foreign ones. But messages can travel through links in both directions regardless whether they are foreign or native to the nodes they are hooked onto.

DANTE's reconnection mechanism is triggered periodically at each node to choose, from a set of candidates C, which other peers that node must point its native connections toward. The reconnection mechanism tries to form topologies that are as centralized as possible, but at the same time prevents any node from becoming congested.

The reconnection mechanism computes the attractiveness Π_i of each candidate peer $i \in C$ as follows:

$$\Pi_i = k_i^{\gamma_i}, \tag{1}$$

where k_i is the degree (number of neighbors) of peer i.

The γ_i value is computed taking into account both the capacity and load of i:

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

$c_{i_{\text{norm}}}$ 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,

$$c_{i_{\text{norm}}} = \frac{c_i}{c_{\max}}, \tag{3}$$

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

$t_{i_{\text{norm}}}$ represents the average time spent by recent searches at node i (time in queue plus processing time), after being 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,

$$t_{i_{\text{norm}}} = \frac{t_i - t_{\min}}{t_{\max} - t_{\min}}. \tag{4}$$

It is straightforward to derive that $0 \leqslant t_{i_{\text{norm}}} \leqslant 1 \forall i$, where a lesser $t_{i_{\text{norm}}}$ means that node i is more attractive, since 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, which is computed as

$$p_i = \frac{\Pi_i}{\sum_{j \in C} \Pi_j}. \tag{5}$$

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

2.3. Candidate sampling

The reconnection mechanism of DANTE depends on the set of candidates C which the node can connect to. There are several mechanisms that could be used to build this list of candidates. For example, a *gossiping*-based service like those presented in [17] could spread information about nodes throughout the network. Another solution is to make nodes keep a *cache* of other peers in the network.

DANTE implements a third solution. Whenever a node i starts a new reconnection, it launches a *Look-For-Node* message that, starting from node i, traverses the network following a random walk with a bounded TTL. When the TTL expires, another message is sent to node i with the list of peers traversed by the *Look-For-Node* message. This list becomes the set of candidates. This technique demands little bandwidth and so has a small incidence on the network load. Moreover, as Newman's results [18] show, the set obtained is a good sample of the overall network. Finally, with high probability nodes with a high degree (potentially the most interesting candidates, as they know more about the resources in the network) will be collected by the message, because they are very likely to be visited by the random walk [18]. This is the same sampling technique used in the previous version of DANTE [16], as already discussed above.

2.4. Reconnection algorithm

To sum up, we present the reconnection process in algorithmic form in Algorithm 1. Recall that the algorithm is executed once the *Look_for_Nodes* message, that carries the list of candidates, arrives back to the node running the reconnection process.

Algorithm 1. Choose N new neighbors from set of candidates C

```
1: Reconnection($C$, degr, capac, times)
    //$C$: Set of candidates
    //degr[]: Candidates degrees ($k_i$)
    //capac[]: Candidates capacities ($c_i$)
    //times[]: Time spent by searches at candidates ($t_i$)
2: if $|C| \leqslant N$ then
3:   return $C$
4: end if
5: $t_{\max} \leftarrow \max\{\text{times}\}$; $t_{\min} \leftarrow \min\{\text{times}\}$; $c_{\max} \leftarrow \max\{\text{capac}\}$
6: // Computing each candidate attractiveness
7: attrac[] $\leftarrow 0$// Attractiveness of nodes ($\Pi_i$)
8: attrac$_{\text{sum}} \leftarrow 0$// Sum of all attractiveness
9: forall $i \in C$ do
10:   $c_{\text{norm}} \leftarrow \text{capac}[i]/c_{\max}$; $t_{\text{norm}} \leftarrow (\text{times}[i] - t_{\min})/(t_{\max} - t_{\min})$
11:   $\gamma \leftarrow 2 \times c_{\text{norm}} \times (1 - t_{\text{norm}})$; attrac[$i$] $\leftarrow$ degr[$i$]$^\gamma$
12:   attrac$_{\text{sum}} \leftarrow$ attrac$_{\text{sum}}$ + attrac[$i$]
13: end for
14: // Choosing new neighbors
15: newNeigh $\leftarrow \emptyset$// New neighbors set
16: while |newNeigh| $< N$ do
17:   // rand($x,y$) returns an uniform random value $z$ ($x \leqslant z \leqslant y$)
18:   randomVal $\leftarrow$ rand(0, attrac$_{\text{sum}}$)
19:   count $\leftarrow 0$
20:   for all $i \in C$ do
21:     count $\leftarrow$ count + attrac[$i$]
22:     if count $\geqslant$ randomVal then
23:       chosenNode $\leftarrow i$
24:       break
25:     end if
26:   end for
27:   $C \leftarrow C - \{\text{chosenNode}\}$; newNeigh $\leftarrow$ newNeigh $\cup$ {chosenNode}
28:   attrac$_{\text{sum}} \leftarrow$ attrac$_{\text{sum}}$ − attrac[chosenNode]
29: end while
30: return newNeigh
```

3. Simulation results

This section shows the results of the simulations performed to study different aspects of DANTE's performance, namely, scalability, tolerance to churn of peers and resilience to attacks.

3.1. Simulation parameters

Each node handles 10 native connections. Reconnections are triggered every 30 seconds. Only five native connections are changed at each reconnection and these connections are chosen uniformly at random from among the 10 native connections of the node. All experiments start with a random topology.

The capacity of each node is set by two parameters: *bandwidth* and *processing capacity*. Each node performs tasks, like processing an incoming message or an internally started process (*e.g.* the triggering of a new reconnection). When performing some task, a node is said to be *busy*. Any other task pending in the node is enqueued until the task that is running has finished.

To compute the amount of time that a node is busy serving some task, two different times are used: the processing time t_{proc} and the sending time t_{send}. The processing time t_{proc} depends on the type of task being processed. If the task does not involve looking for a resource in the list of known resources then $t_{\text{proc}} = 1$. Otherwise, t_{proc} is proportional to the number of resources checked, m, and the node's processing capacity c_i, as $t_{\text{proc}} = \frac{m}{c_i}$. The sending time t_{send} represents the time required by the task to send a message. If no message is sent, then $t_{\text{send}} = 0$. Otherwise, t_{send} depends on the node's bandwidth b_i and the packet size s, as $t_{\text{send}} = \frac{s}{b_i}$. Finally, the time the node is busy is computed as $\max\{t_{\text{proc}}, t_{\text{send}}\}$. This time is not $t_{\text{proc}} + t_{\text{send}}$ as we assume that the sending of messages and the processing of tasks run in a pipeline.

Unless explicitly stated, in all the simulations the nodes' capacities and bandwidths are assigned following the distribution depicted in Table 1. This distribution is derived from the measured bandwidth distributions of the Gnutella nodes reported in [19]. c_i is expressed in resources processed per microsecond, and b_i is expressed in bits per microsecond.

The load on the system is due to the resource lookups initiated by the network peers. The time each node waits between starting two searches is not fixed. Instead, after initiating a search, the node computes the time until the next lookup using an exponential distribution. The mean of this distribution is set by a parameter called the *time between searches*, *tbs*. So, for example, if the network has 10,000 nodes and *tbs* is set to *tbs* = 5 s, then, on average, 2000 searches are started during each second of the simulation.

Table 1
Capacity and upload bandwidth distribution used in the simulations

Percentage of nodes (%)	Processing capacity c_i	Bandwidth b_i
20	0.1	0.01
45	1	0.1
30	10	1
4.9	100	10
0.1	1000	100

In all experiments, each node holds 100 resources, all of which are different. All resources are equally popular, *i.e.* all resources have the same probability of being the resource looked for by each new search. Moreover, the replication of resources is *uniform*, since the number of copies of each resource available in the system is the same for all resources. This is one of the hardest setups for any resource location algorithm, as shown in [20]. In that work, Cohen and Shenker show how *uniform replication*, along with *proportional-to-popularity replication*, are the extreme cases of a family of replication strategies of which they both have the largest expected search length, and so the poorest performance. This holds independently of the popularity distribution. We deem that replication strategies outside this family, on the other hand, are unrealistic in real-world networks.

Another work [11] from Lv et al. follows the line of [20]. In their experiments they compare different combinations of strategies for resources popularity and replication. They find out that the worst combination in terms of load due to search messages is the uniform popularity with uniform replication, the combination we implement. To get a high success rate (as we intend) this combination produced a much higher load on the system than other strategies as searches required more hops to be solved. For example, the Zip-f popularity distribution (that is known to be present in many real world networks [9]) combined with the proportional to popularity replication strategy (the most 'natural' replication), achieved much better results. Searches required less hops to find resources and so the load on the system was also lessened.

The TTL of the *Look_for_Nodes* message was set to 30 (we checked empirically that this value was enough to get a good sampling of the network). The TTL of the search messages, on the other hand, was set to 1000. With this setting we aimed to ensure a high success rate (as we will see in the next sections, we reached a success rate very close to 100%) although this could introduce more load due to search messages on DANTE.

3.2. Adaptability and robustness

The simulations presented in this section aim to test the adaptive behavior of DANTE, that is, how the topology evolves to form efficient configurations.

As the topologies built by DANTE are often centralized, it could be argued that DANTE is vulnerable to attacks targeted at well-connected nodes. Thus, in the simulations run for this section we also check how DANTE's self-adaptation mechanism reacts to such events. At minute 30, when the network has already moved from the initial random topology to a centralized one, a targeted attack is simulated: the 10 best-connected nodes (the central nodes) leave the network. When this occurs, the searches waiting in their queues are *discarded*. Additionally, their neighbors change their connections by pointing them toward some other peers which are chosen at random from the remaining peers. Thirty minutes later, the attacked nodes are reactivated. We will examine how DANTE reacts to such events.

The results are shown in Figs. 2–5. All of them show how network behavior evolves in the first 90 min of virtual time. The simulations presented here were run with $10,000$ nodes, and replication $r = 0.01$. Two different loads were tested, $tbs = 5$ and $tbs = 2.5$ s.

First we study the effect of DANTE's reconnection mechanism on the topology. As discussed before, the reconnection mechanism tries to form centralized topologies where the most capable nodes are also the ones with a higher number of connections.

To study the topology evolution we use the *network Clustering Coefficient* [21] (CC) as a metric that shows the degree of centralization of the network. Formally, it is defined as follows. Let $G = (V, E)$ be an undirected graph without loops nor multilinks, where vertices V represent

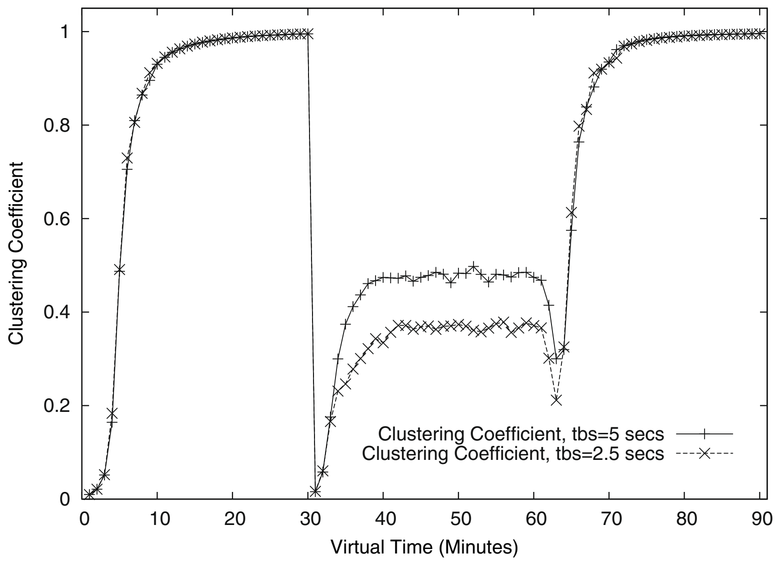

Fig. 2. Adaptability and robustness – clustering coefficient.

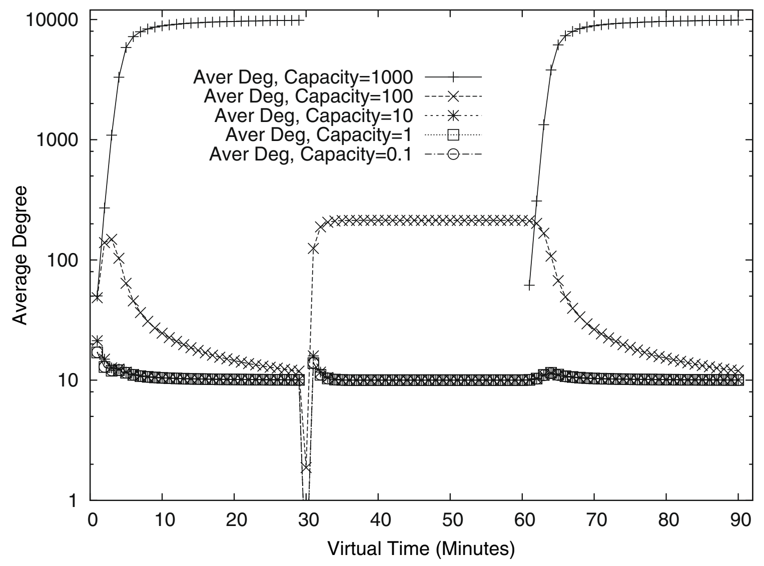

Fig. 3. Adaptability and robustness – average degree by node capacity.

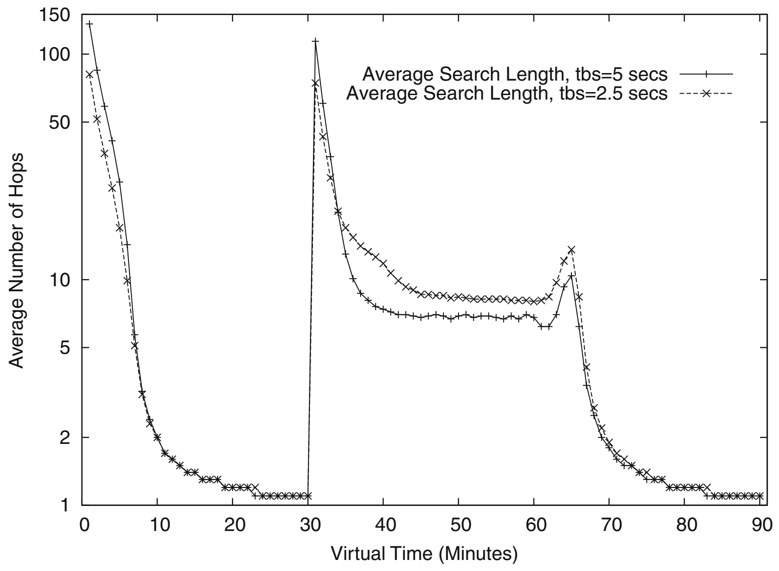

Fig. 4. Adaptability and robustness – search length under attack.

the nodes and edges $E \subseteq V \times V$ are the links between nodes (if nodes i and j are connected, then $(i,j) \in E$ and $(j,i) \in E$). Let N_i be the set of neighbors of node $i \in V$, $k_i = |N_i|$. Then the clustering coefficient of node i, CC_i, is defined as the number of connections among i's neighbors, divided by the maximum amount of possible links among them:

$$CC_i = \frac{|(N_i \times N_i) \cap E|}{k_i(k_i - 1)} \tag{6}$$

and the clustering coefficient of the network, CC, is given by the expression:

$$CC = \frac{1}{|V|} \sum_{i \in V} CC_i. \tag{7}$$

It is straightforward to observe that $0 \leqslant CC \leqslant 1$. The closer CC is to 1, the more centralized the topology of the graph is. Conversely, the closer CC is to 0, the more randomized the topology is.

Fig. 2 shows how the clustering coefficient of the network changes as the virtual time passes for the two loads tested, $tbs = 5$ and $tbs = 2.5$ s. Initially, the CC value is quite low as the network starts from a random topology. But almost immediately, after a few reconnections, the clustering coefficient grows until it reaches values very close to 1. DANTE's reconnection process has shifted the topology to a centralized state, where all peers are connected to (and only to) a small set of central peers. We note that very similar topologies are achieved with both loads

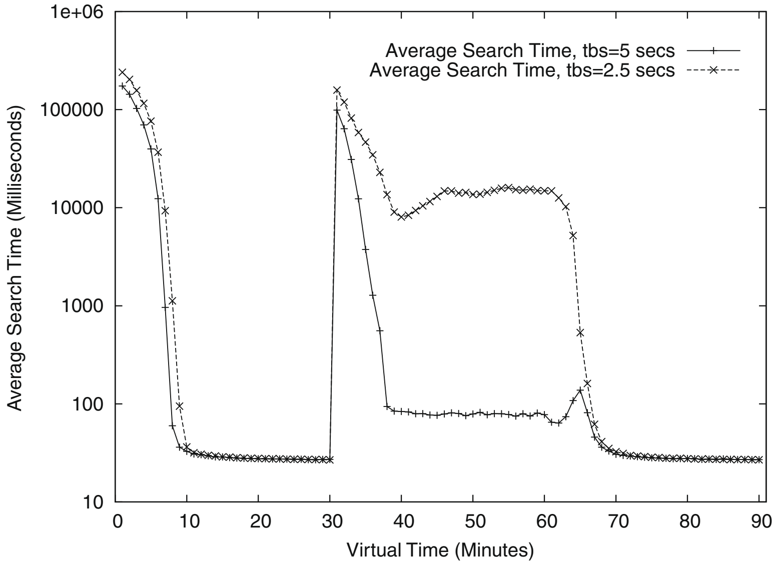

Fig. 5. Adaptability and robustness – search time under attack.

during the first part of the simulation, although the greatest load seems to make the network evolve slightly faster. The reason is that a higher load makes the most capable peers even more attractive to the reconnection process as compared with the weaker node.

Later, when the attack is performed at minute 30 on all the central nodes, the remaining peers redirect their connections randomly so a random topology appears again. From that moment on, nodes will try to connect to the remaining peers with a higher capacity (nodes in the fourth row in Table 1). The topology does not reach a star-like form again, as there are no peers with enough capacity to become central nodes. Yet, highly connected nodes (*hubs*) appear, so the clustering coefficient grows again in a few minutes. Also, we can see that the topology built after the attack is more centralized for the lesser load. This is because the adaptation process takes into account the saturation at the nodes. With $tbs = 2.5$, even the most capable of the remaining nodes cannot handle as many connections as with $tbs = 5$. Thus, the resulting topology is less centralized as those nodes will have fewer links.

Finally, when the attacked nodes are back at minute 60, the network changes to a star-like topology again. To form that topology, nodes disconnect from the present hubs and point their native connections to the just arrived peers. Because of this, we observe that the clustering coefficient is diminished as the network gets more 'randomized' due to these changes, although this happens for a small range of time. Almost immediately the *CC* raises again until a value close to 1. Here, we also see how the adaptive process of DANTE has a very desirable property: it always tries to adapt so as to achieve the best possible topology using the most capable nodes available at any time.

We have stated so far that the most capable nodes are the ones to get the highest degrees. Fig. 3 confirms this. It shows, for the experiment with $tbs = 5$ s, the average degree of nodes grouped by their processing capacities, and how this average degree changes as the virtual time passes. We see that already after the first reconnections (results at minute 1) the nodes with processing capacities 1000 and 100 get more links that the rest of peers. After a few reconnections more, the nodes with the greatest capacity get an average degree very close to the maximum number of connections possible (*i.e.* the size of the network), while the other peers only keep their native connections. This means that the highest capacity nodes are connected to all the other peers in the system, so they have become the central nodes of the star-like topology. Immediately after the central nodes are attacked at minute 30, the nodes with capacity 100 quickly recollect again many links, in some way 'replacing' the central nodes and becoming hubs of the system. However, they are not able to handle as many connections as the peers with capacity 1000. Also note that the weakest nodes keep having the minimum degree. Finally, when the most powerful nodes are back at minute 60, they soon get many connections until they become central nodes once more.

To study the topology performance we focus on the search results. The *average search length* is the average number of hops performed by a random walk to find a resource. When the topology is centralized, the average search length is 1 (all search messages go to the central nodes, which can reply to all queries since they know all the resources in the system). Moreover, the less centralized the topology is, the longer the search length will be. The *average search time*, on the other hand, allows us to study how topology affects network efficiency. As expected, our results show that the more centralized the network is, the less time is required by the searches.

Fig. 4 shows how, during the first minutes of the simulation, the average search length decreases sharply after a few reconnections, until it reaches a value close to 1 as the network moves to a star-like topology. At minute 30, because of the attack on the central nodes, the network moves again to a random topology. Due to this, searches again require many hops to find resources. As new reconnections are performed and hubs appear the average search length is decreased once more. But the new topology is not centralized, so it can not be as low as before. This only happens when the attacked nodes return to the network and DANTE builds again a centralized topology. Also we observe that between minutes 30 and 60 the length of searches is lesser for $tbs = 5$ s than for $tbs = 2.5$ s. This agrees with Fig. 2, that shows how the topology achieved with $tbs = 5$ during that time interval is more centralized that the one built with $tbs = 2.5$.

In Fig. 5 we can see the effect of the attack on the average search time. After the attack this time is increased to values close to those obtained at the beginning of the experiment. This is to be expected, since the topology after the attack is again random. Then, as nodes change their connections, the topology is adapted to a clustered form, thus lowering the average search time to lower values. Finally, when the 10 nodes attacked are back, the search times gradually return to the values prior to the attack, as the system adapts itself to form a centralized topology again.

From Fig. 5 it seems straightforward to conclude that a more centralized topology achieves better performance, which confirms the analysis of Guimerà that inspired DANTE. It is important to note, however, that this is also achieved because DANTE's reconnection mechanism avoids congestion on peers, so no node gets more connections that it can handle. To accomplish both this and the goal of achieving centralized topologies, DANTE ensures that the nodes with the greatest capacities are also the ones that get the highest number of neighbors.

The proportion of discarded searches (searches in the queues of the attacked nodes) is around 0.005% of the total in both experiments. The proportion of failed searches is less than 0.04%. 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 were not present for 30 min. Therefore, searches run during that interval to look for the resources in those nodes could not finish successfully.

From the experiments outlined here we conclude that DANTE succeeds in building efficient topologies where searches require little time to be solved, while also avoiding congestion at well-connected nodes. DANTE is also able to take advantage not only of the capabilities of the nodes already present but also of those that arrive at the system, by always trying to build topologies that offer better performance (that is, it does not get 'stalled').

We can also infer that the effects of attacks on DANTE depend on factors like which nodes are attacked or the load on 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 so as to reach another efficient state again. *The system is never, and can never be, fully shut down by only attacking a subset of its peers (however big this subset is), because it is not dependent on any particular group of nodes.* It is true nonetheless that under some topologies a well-directed attack could partition the network, specially topologies with a low *conductance* [22]. Intuitively, these are topologies where big regions of the network are connected by few nodes. However, due to the randomness applied by the reconnection heuristic, DANTE tends to form topologies where such regions do not appear as nodes can potentially connect to any other node in the network. This is reinforced by the fact that nodes in DANTE, when some neighbor is attacked, redirect the corresponding connection to some other peer chosen at random. A way to implement this could be to use a cache of known peers, from the results of the last *Look_for_Node* messages.

3.3. Scalability

An important concern when working with P2P systems is scalability. The system should be capable of managing thousands of nodes, while keeping the performance within acceptable levels. In this section we present the results of simulations that show how, in DANTE, overall performance can even increase when new nodes are added.

The simulations in this section were run with five different network sizes: 2000, 4000, 6000, 8000, and 10,000 nodes. Nodes' capacities are set using the proportions depicted in Table 1. Only results related to searches started between minutes 31 and 60 (inclusive) of virtual time are used. When all searches that started before minute 61 have finished the simulation is stopped. Two replication rates are used, $r = 0.05$ and $r = 0.1$, each with two different loads $tbs = 5$ and $tbs = 2.5$ s.

On observing Fig. 6 we detect a somewhat surprising effect: the average search time *decreases* as the network grows. This is due to the fact that, given the proportions in Table 1, the bigger the network is, the more nodes with high capacities are added to the system. This leads to better performance, as those nodes can manage many connections and hence they allow forming topologies where the average number of hops needed to solve searches is decreased. This is shown in Fig. 7.

We thus see that DANTE is able to manage large network sizes properly, due to its ability to take advantage of the high-capacity nodes present in the network.

3.4. Performance under churn

Peers in real networks can enter and leave the network at a high frequency and, therefore, the system must be able to manage a high transitivity of peers. In this section we will study how DANTE behaves under different churns of peers.

In all the simulations run for this section, the probability of each node being active at start time is 0.5. For each active node calculations are performed to determine how long they will remain active before leaving the network. This time is computed using an exponential distribution with a certain average value, set as a parameter of the experiment called the *average active time*. The lower that value is, the higher the churn of peers will be. When the time to be active expires for a particular node, then that node is deactivated: it discards all searches in its queue and closes its connections with all its neighbors. After

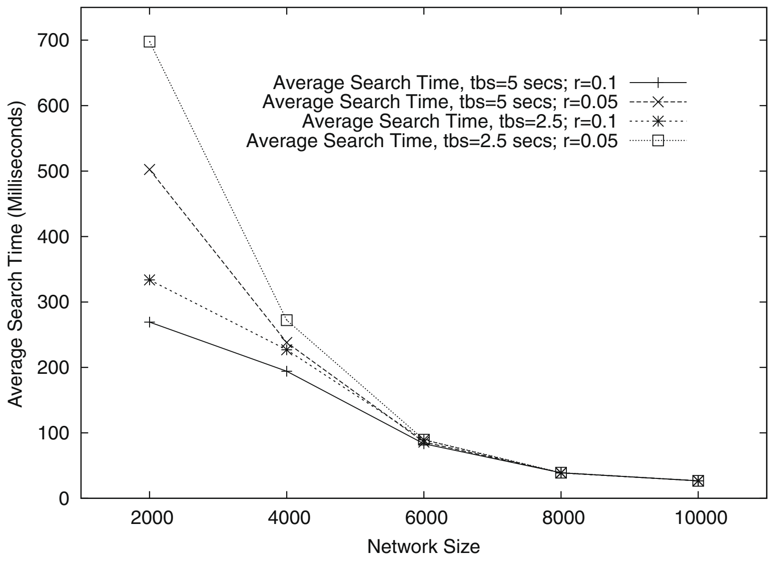

Fig. 6. Scalability – search time as network grows.

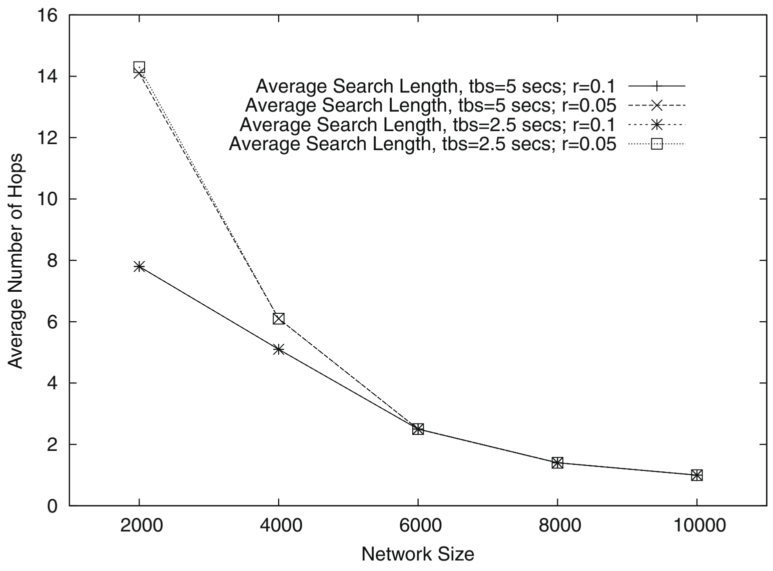

Fig. 7. Scalability – search length as network grows.

0.5 s of virtual time, the node is reactivated and reenters the network, connecting its *native* connections to 10 nodes chosen randomly from among the rest of the active peers. The time it will remain active is computed again. If any node has a *native* connection pointing toward a peer that leaves the network, it will redirect that connection to some other active peer chosen at random. This works as a 'provisional' solution until a new reconnection is run. This setting simulates a scenario with nodes entering and leaving the network simultaneously (a similar approach is used in [15]).

The following values were used for the *average active time* parameter: 60, 300, 600, 3000, and 6000 s. Two different replication rates r were applied: $r = 0.05$ and $r = 0.1$. All the simulations run for this section used a network size of 10,000 nodes. Only searches started between minutes 31 and 60 (inclusive) were taken into account in order to compute the final results.

The first result we study, shown in Fig. 8, is the proportion of searches that were *successful*, *failed* (the search message TTL expired before the resource was found) or *discarded* (a search is discarded if it is in the queue of a node that leaves the network). The load was set to $tbs = 5$ s for these experiments. As Fig. 8 shows, for the maximum churn, when the average active time is set to 60 s, the sum of failed and discarded searches is about 4.6% of all searches for $r = 0.1$, and 6.2% for $r = 0.05$. We deem this to be a small impact for such a high transitivity.

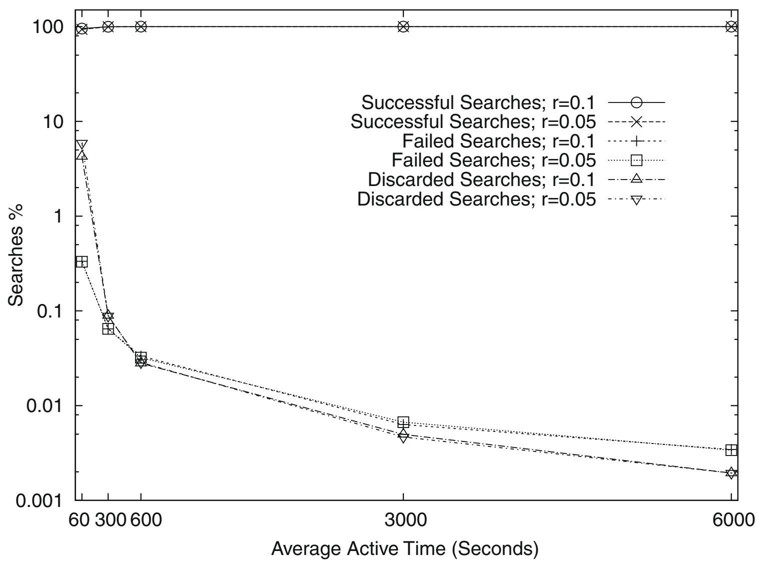

Fig. 8. Churn – searches count (in%) under churn.

Moreover, with a small increase in the stability of nodes so that their average active time is 300 s, the number of failed and discarded searches is drastically reduced to less than 0.2% of the total, a proportion that goes on decreasing as the average active time grows.

Next, we comment on how churn affects the average length of searches. A high transitivity of peers prevents the network from achieving centralized topologies since, in our experiments, all nodes have the same average active time regardless of their capacity (and therefore their attractiveness). Thus, nodes that have become well-connected will leave the network over and over again. It is reasonable to think that in real networks some of the most powerful nodes will tend to be more stable, so centralized topologies can be built (sometimes nodes might join the network precisely for that purpose, as for example the superpeers in the eDonkey [23] network). However, we have chosen not to make such an assumption in order to test DANTE in a more difficult scenario.

As we can see in Fig. 9, the higher the churn is, the less centralized the topology achieved by DANTE will be. But even for the highest transitivity of nodes, DANTE is able to form topologies with nodes very well connected, although eventually these peers leave the system thus forcing the rest of the nodes to adapt the topology. In addition, we can see that increasing the churn of peers has little effect on the search length for a wide range of values. We have to reach very small values of the average active time to

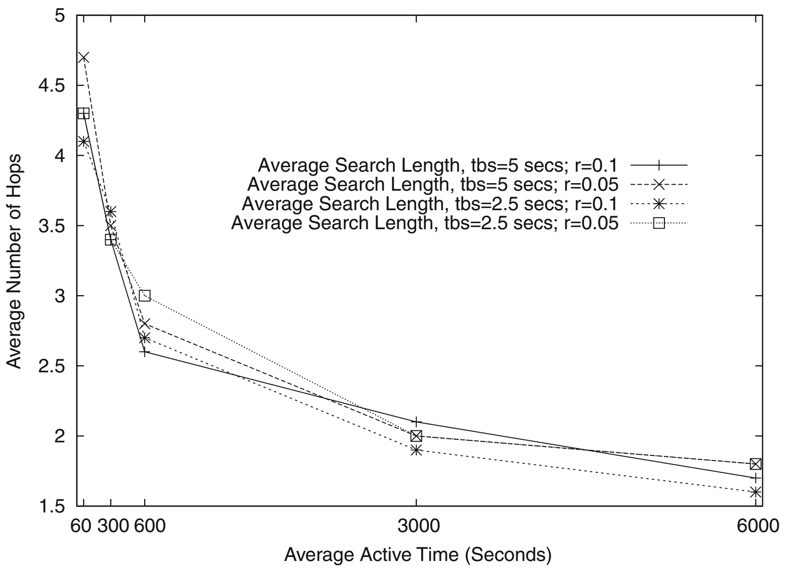

Fig. 9. Churn – search length under churn.

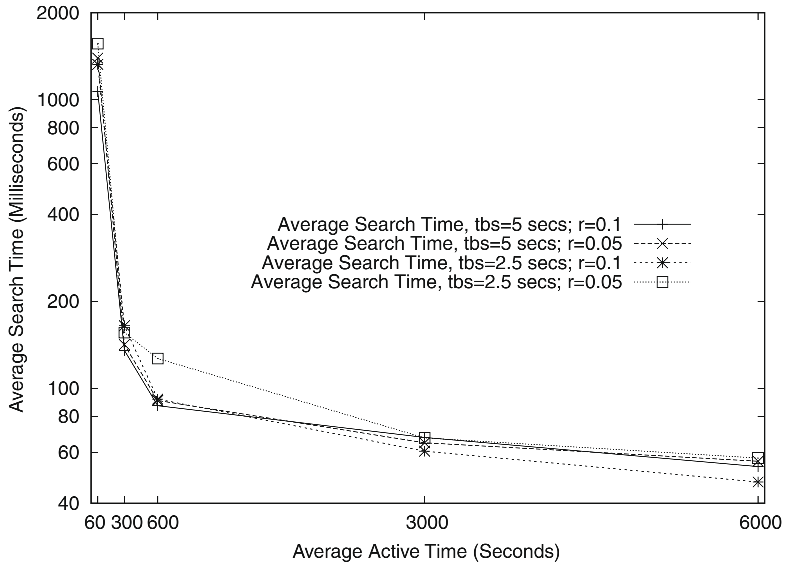

Fig. 10. Churn – search time under churn.

see a (relatively) strong increase in the search length. This is true for the two loads tested, $tbs = 2.5$ s and $tbs = 5$ s.

Finally, in Fig. 10 we can observe the impact of the transitivity of nodes on the average search time. It shows the results of the same experiments used for Fig. 9, but this time referring to the average duration of searches. For low churns, the search time behaves in a similar way to the search length, with a smooth growth until a very low average active time is reached. It is only at this point that a sharp increase in the search duration can be seen. Note that this growth is not linear with the changes in the average search length. This is due to the fact that small increments in the search length have a high impact on search times, which is a problem that cannot be avoided. The reason is that, as searches follow a pure random walk, a higher search length implies that low capacity nodes will be traversed more frequently by search messages. In turn, this makes searches to spend more time on the system before being solved. However, DANTE succeeds in building topologies with well-connected nodes that keep the search length bounded. Thanks to this, only very high churns can effectively raise the average search duration.

From the results shown in this section, we can infer that only a high transitivity of peers (regardless of their capacity) can have an impact on DANTE's performance, although that impact is limited. DANTE is able to form close-to-centralized topologies that keep the average search length and duration bounded, even when the system has little stability.

3.5. Comparing DANTE and Gia

As explained in Section 1, Gia is another proposal for a P2P system that uses an adaptation mechanism to improve the efficiency of searches. In [15] the authors of Gia carried out several simulations that show how self-adapting networks can offer better performance than other solutions (like flooding) in a variety of scenarios. Thus, instead of repeating those same simulations with DANTE, we considered it more interesting to compare Gia and DANTE. In order to do so, we developed a Gia simulator that implements the mechanisms described in [15] that constitute the basis of their system: a *flow control* system to avoid overloading nodes, a *biased random walk* search mechanism, and a *topology adaptation* protocol.

It is important to note that, although both Gia and DANTE implement a topology adaptation mechanism, they are in fact different systems. For example, DANTE does not use any flow control technique to avoid overloading nodes; instead, its reconnection mechanism reduces the number of links to congested peers. But the key difference is that DANTE actively seeks to form centralized topologies, an approach that differs from Gia. As commented in Section 1, nodes in Gia use a magnitude called the *level of satisfaction*, which basically measures the difference between the capacity of the node and the capacity that its neighbors devote to it. Gia nodes seek to balance their level of satisfaction and, as a result, high-capacity nodes tend to have more connections than the rest, thus forming topologies where the search length is kept low. The DANTE approach, on the other hand, is different. Its main purpose is to form topologies that are as centralized as possible and it pursues this objective almost explicitly by means of its definition of attractiveness (see Section 2.2). The goal of the experiments in this section is to compare the two adaptation techniques. As we will see, DANTE achieves better performance because it is able to form more efficient topologies.

Simulations were run with 1000 nodes, with a replication $r = 0.1$. As usual, nodes capacities and bandwidth were set following the distribution shown in Table 1. Only searches that started between minutes 31 and 60 were taken into account. As in the previous experiments, the load in each experiment was generated by the resource lookups started by the peers. This load was set, as before, by the *time between searches* (*tbs*) parameter. Six experiments were run for each proposal, each one with a different *tbs*.

In Fig. 11 we plot the average search times for the 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. Fig. 12 helps us to understand the reason for DANTE's better behavior: searches in Gia need a far greater number of hops to find a certain resource (about 160) than in DANTE (about 7). The reason for this is that, although the topology in DANTE is not totally centralized (as there are not enough high-capacity nodes), it still maintains 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, does not form such a topology, as this is not its goal. In Gia each node tries to get a set of neighbors with a sum of spare capacities that matches its own. That is, in a way it aims to 'get' from their neighbors as much capacity as they devote to them. Because of this, Gia's reconnection mechanism forms topologies much more randomized than DANTE even for low loads. This forces searches to perform more hops to find resources, which has an important impact on the system's performance.

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

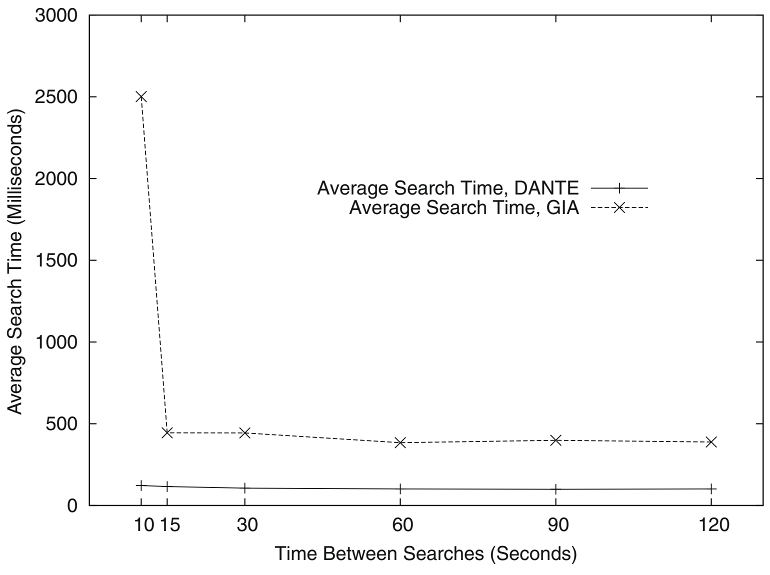

Fig. 11. Gia and DANTE search time.

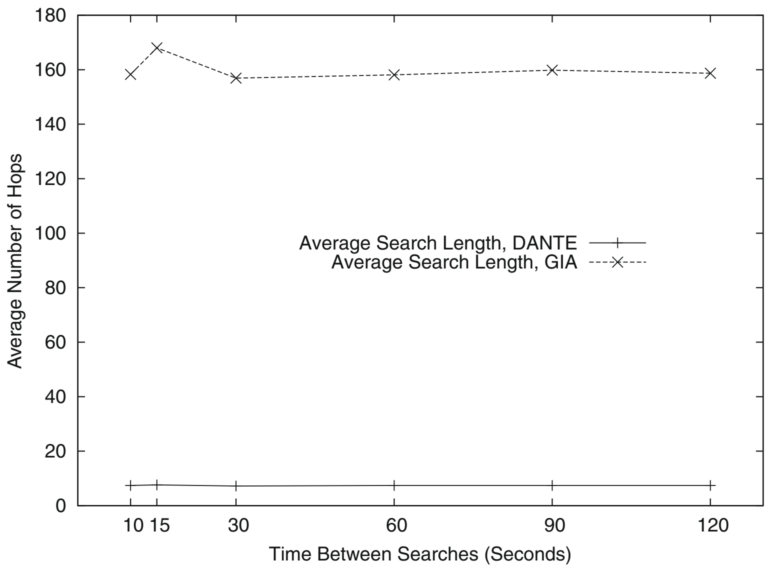

Fig. 12. Gia and DANTE search length.

Thus, we can conclude that the DANTE approach can lead to more efficient topologies, where searches are solved in fewer hops and less time.

4. Conclusions

P2P systems are a promising paradigm, yet they demand innovative solutions to new problems, like decentralized resource location. With DANTE we propose a self-adapting mechanism that forces the network to change its topology with the aim of achieving an efficient configuration that depends on the system load and the peer capacities. The results obtained with DANTE and presented in this article seem promising. However, much work remains to be done in order to improve the performance of these techniques.

Future work. There are some aspects of DANTE that we consider worthy of further exploration, for example, other ways to sample the set of candidate nodes *C*. It could also be interesting to study heuristics to decide when to trigger the reconnection mechanism: if the network has reached a stable state, maybe is not worth running new reconnections until some event such as an attack occurs. Another line to study is how to tune the TTL value to optimize the performance. Finally, new reconnection heuristics could be studied and developed.

References

[1] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM Computing Surveys 36 (4) (2004) 335–371.
[2] J. Risson, T. Moors, Survey or research towards robust peer-to-peer networks: search methods, Computer Networks 50 (17) (2006) 3485–3521.
[3] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: a global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (2004) 41–53.
[4] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, 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.
[5] B.F. Cooper, An optimal overlay for routing peer-to-peer searches, in: Lecture Notes in Computer Science, Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference, vol. 3790, Springer-Verlag, Grenoble, France, 2004, pp. 89–101.
[6] A.I.T. Rowstron, 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.
[7] Q. Lv, S. Ratnasamy, S. Shenker, Can heterogeneity make Gnutella scalable? in: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Cambridge, United States, 2002, pp. 94–103.
[8] The gnutella website. <http://www.gnutella.com>.
[9] K. Sripanidkulchai, The popularity of gnutella queries and its implications on scalability, 2001, in O'Reilly's P2P Open Conference. <http://www.openp2p.com>.
[10] G.H.L. Fletcher, H.A. Sheth, K. Borner, Unstructured peer-to-peer networks: Topological properties and search performance, in: Lecture Notes in Computer Science (Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing), vol. 3601, Springer-Verlag, New York, New York, United States, 2004, pp. 14–27.
[11] Q. Lv, P. Cao, E. Cohen, K. Li, 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, 2005, pp. 84–95.
[12] L.A. Adamic, B.A. Huberman, R.M. Lukose, A.R. Puniyani, Search in power law networks, Physical Review E 64 (2001) 46135–46143.
[13] R. Guimerà, A. Díaz-Guilera, F. Vega-Redondo, A. Cabrales, A. Arenas, Optimal network topologies for local search with congestion, Physical Review Letters 89 (2002) 248701.1–248701.4.
[14] V. Cholvi, V. Laderas, L. López, A. Fernández, Self-adapting network topologies in congested scenarios, Physical Review E 71 (3) (2005) 035103–103 5103-4.
[15] Y. Chawathe, S. Ratnasamy, N. Lanham, 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, 2003, pp. 407–418.
[16] L. Rodero-Merino, L. López, A. Fernández, V. Cholvi, Dante: a self-adapting peer-to-peer system, in: Lecture Notes in Computer Science (Proceedings of AP2PC 2006, to be published), Springer-Verlag, 2006.
[17] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, M. van Steen, The peer sampling service: experimental evaluation of unstructured gossip-based implementations, Lecture Notes in Computer Science (Proceedings of Middleware 2004), vol. 3231, Springer-Verlag, 2004, pp. 79–98.
[18] M.E.J. Newman, A measure of betweenness centrality based on random walks, Social Networks 27 (2005) 39–54.
[19] S. Saroiu, P.K. Gummadi, 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.
[20] E. Cohen, S. Shenker, 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, 2002, pp. 177–190.
[21] S.H. Strogatz, D.J. Watts, Collective dynamics of small-world networks, in: Nature, vol. 393, 1998, pp. 409–410.
[22] R. Kannan, S. Vempala, A. Vetta, On clusterings: good, bad and spectral, in: Journal of the ACM, vol. 51, 2004, pp. 497–515.
[23] The eDonkey website <http://www.edonkey2000.com>.

Luis Rodero-Merino is a Junior Researcher at Telefonica I+D in Madrid, Spain, since 2008. He graduated in Computer Science at Valladolid University, and received his PhD degree in Computer Science at Universidad Rey Juan Carlos in 2007, where he also worked as an Assistant Professor before joining Telefonica I+D. Previously he had worked in the Research and Development area of Ericsson Spain, where he developed services for fixed and mobile telephone networks. His research interests include computer networks, distributed systems, P2P systems and grid computing, in particular the study of overlay-networks based solutions and SLA protection.

Luis Lopez is an Assistant Professor at Universidad Rey Juan Carlos in Madrid. His current research work is concentrated on the applications of novel mathematical paradigms, like Game Theory or Complex Systems Theory, into the field of computer communications. He has coauthored more than 40 research papers in different scientific journals and conference proceedings within this topic. He obtained a Telecommunication Engineering degree at Universidad Politecnica de Madrid and at ENST – Telecom Paris in 1999. He also posses a PhD degree in Computer Science by Universidad Rey Juan Carlos since 2003.

Antonio Fernandez Anta is a Professor at the Universidad Rey Juan Carlos in Madrid, where he has been on the faculty since 1998. Previously, he was on the faculty of the Universidad Politecnica de Madrid. He graduated in Computer Science from the Universidad Politecnica de Madrid in 1991. He got a PhD in Computer Science from the University of Southwestern Louisiana in 1994 and was a postdoc at the Massachusetts Institute of Technology from 1995 to 1997. He is currently Senior Member of IEEE and ACM.

Vicent Cholvi graduated in Physics from the University of Valencia, Spain and received his doctorate in Computer Science in 1994 from the Polytechnic University of Valencia. In 1995, he joined the Jaume I University in Castellon, Spain where he is currently an Associate Professor. His interests are in distributed and communication systems.