Performances Study of the Distributed Spanning Tree an Overlay Network for Server Lookup

Sylvain Dahan, Alexandru Dobrila, Jean-Marc Nicod, Laurent Philippe

Performances Study of the *Distributed Spanning Tree* an *Overlay Network* for server lookup

Sylvain Dahan, Alexandru Dobrila, Jean-Marc Nicod and Laurent Philippe
Laboratoire d'Informatique de l'Université de Franche-Comté
16, Route de Gray - 25030 Besançon cedex - France

Abstract

Random graph and tree are two topologies used to build overlay networks. These overlay networks may be used by large scale discovery mechanisms to run search algorithms. The Distributed Spanning Tree (DST) is another topology that may be used as overlay. In a DST, every computer is a leaf. DST's non-leaf nodes are sets of computers instead of computers. Thus, it allows the use of tree traversal algorithms while avoiding the usual tree's bottlenecks. As a result, the DST allows more efficient executions of search algorithms in term of number of sent messages and in term of load balancing. In this paper, we describe the results of several simulations of flooding algorithm executions. These simulations are run on the three previous topologies and for different numbers of nodes. These simulations indicate that the DST structure is more efficient than graph topology which, in turn, is more efficient than tree topology in term of traversal speed and in term of supported load for every simulated scale. We study as well the behaviour of the DST when nodes are added or deleted to show that the structure is adapted to dynamic environments.

1. Introduction

Grid middleware allows resources sharing between several entities. These resources consist into a wide variety of hardware, data and software. As the size of Grid deployments increases, finding the resource that is needed by a user becomes an issue. To solve this issue, peer-to-peer systems are an inspiration source for Grid middleware because they propose several scalable search mechanisms [8]. These mechanisms could be classified into two categories [9].

The first category is composed by mechanisms based on a directory. Each resource has to register itself by specifying its name and its location to be visible. Then, clients can find this resource by giving the corresponding name to the directory. Distributed Hash Tables are a good example of directories based peer-to-peer search mechanisms.

The second category is composed by mechanisms based on flooding algorithms. Information on resources are stored on each computer participating to the peer-to-peer system. The computers are connected together to form a communication graph, also called overlay network. When a peer, a node of the graph, is looking for some information that it does not already have, it asks its adjacent nodes for the information. If they do not own it, these nodes forward the request to their adjacent nodes, and so on until the information is found or until a specified maximum number of hops is reached (usually called Time To Leave or TTL).

The performances of flooding algorithms depend on many factors. One of them is the topology of the communication graph. The tree is an interesting topology because its complexity in number of exchanged messages is optimum as it directly depends on the number of nodes. But searches on trees generate bottlenecks when the load of the system increases. So flooding algorithms are run most of the time on pseudo-random graphs although their complexity in term of number of messages is higher.

To improve the performances of flooding algorithms, we propose a new way to build a tree which does not have the usual tree's bottlenecks. This original structure, called Distributing Spanning Tree (DST), has already been described from a theoretical point of view in two previous publications [1, 3]. But no study about how it behaves in practise has been made public. The purpose of this paper is to share pertinent results and comments about simulations that were made to study the performances of flooding algorithms using a DST structure. These performances are compared with two others topologies: pseudo-random graphs and trees.

Simulations have been used numerous times to study performance of flooding algorithms in various contexts. For example, G. Fletcher and H. Sheth [5] compare search on a CAN distributed hash table, on a random graph and on a Pandurangan Graph to evaluate their efficiency . L. Qin *et Al.* [10] wrote a comparative study about performances and supported load of various random-graph topologies and various resources distribution. Beside simulations, some peo-

ple, like R. Gaeta *et Al.* [6], also propose analytical model for those search algorithms on random graphs.

The article is organised the following way. The DST is shortly presented in section 2. We study the cost in term of messages when nodes arrive or leave the DST in section 3. The performances comparisons are conducted on a simulator described in 4. The simulations results are discussed in section 5 and exhibit the interest of the DST structure for search algorithms. Then, we conclude on the use of the DST structure in several distributed contexts.

2. The DST Structure

The DST structure is designed to correct the drawbacks of trees and random graphs in large scale overlay networks.

The DST is a virtual *tree* which covers the whole set of computers — *spanning* — and where nodes are *distributed* across several computers to avoid bottlenecks. A DST is similar to a B-Tree [7]: it is composed of several levels. All leaves belong to the same level, each non-leaf node has a limit number of sons and the root, if it is not a leaf, has at least two sons. The level 0 contains the leaves of the tree. The level 1's nodes are complete graphs of leaves,

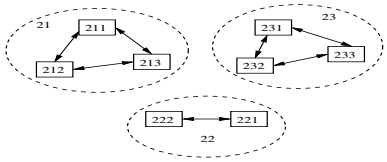


Figure 1: Level 1's nodes of a DST

Fig. 1 presents 3 nodes of level 1 and 8 leaves. Upper levels of a DST are structured the following way: any node of level *n+1* is a complete graph of its children which are nodes of level *n*. The Fig. 2 is an example of a level 2's node.

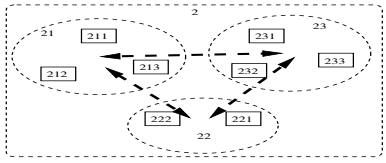


Figure 2: Level 2's nodes of a DST

By definition, a DST's node is distributed through its descendants. Thus, a DST's node is distributed through the computers that are leaves of the node's subtree. To implement a node of level *n* with *n>2* we connect each computer of this node to at least one computer of each children of the node as shown in Fig. 3. Thus, any computer of a node can send a message to every children of this node via the computers that are inside these children.

Using this construction, each node generates its own "routing table". The routing table of one node has as many

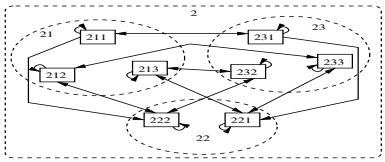


Figure 3: Links that implement a complete graph of graphs.

entries as levels in the DST and each entry contains a link to a computer that resides inside each brother nodes. On Fig. 3, if node 2 wants to broadcast a message to its descendant, it will send a message to node 21, node 22 and node 23. If, the process that initiates the broadcast is located in node 213, it will send its message to the process of node 21 located in node 213, to the process of node 22 located in node 221 and to the process of node 23 located in node 232. Then, node 21, node 22 and node 23 forward the message to their children using their complete graph.

This quick explanation of the DST's structure provides enough information to understand that a DST is a tree where non-leaf nodes are complete graphs of their children. By recursively creating complete graphs, we are able to distribute non-leaf nodes between their children and create a spanning tree per node. As every computer acts as a node of different levels, it is possible to share the load of these levels between several and thus avoid the usual tree's bottlenecks. For more information about the DST structure and details on construction algorithms, a complete description of the DST structure is available in [3].

3. The DST management

The DST has an incremental structure : we add and we delete nodes as participants come or leave the DST. To be a DST member, a node must know another node which is already in the DST. We study here the cost of adding and deleting nodes from the DST. We also introduce the notion of limits with an inferior and superior limit which apply to the size of a node. If a node is too small (under the inferior limit) it should be merged, it should be split if it is too big (above the superior limit).

3.1. Cost of adding nodes

Figures 4 and 5 show the number of messages for a 4 and a 8 levels DST. The abscissa indicates the number of messages necessary to add a node and the ordered indicates the number of nodes that has generated this number of messages (two nodes can generate the same number of messages). In this figure, each peak matches with the split of a higher level node. If the node is high in the hierarchy then more nodes will be affected by the splitting which implies an important number of messages. These figures indicate that the number of messages needed to split a node

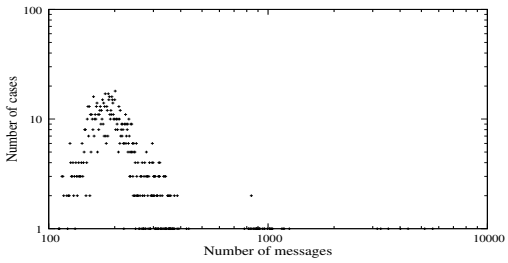


Figure 4: Number of messages with 4 levels.

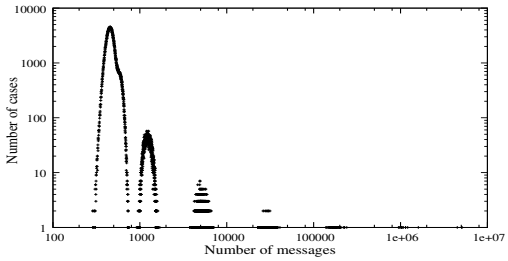


Figure 5: Number of messages with 8 levels.

grows exponentially with the level of the node. Besides, the number of splits for a node decrease logarithmically with its level in the DST.

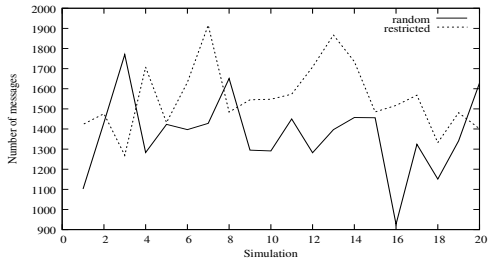
Thus, we have a lot of splits into first levels, those are costless in comparison with higher level splits which are far more seldom. In this simulation we have a relation of ten between both.

3.2. Cost of deleting a node

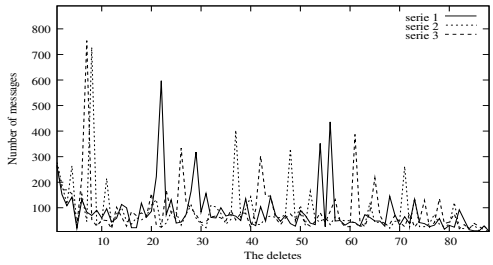
We study here the impact of different factors coming from the departure of a node.

Impact of the area where the node is deleted We test the DST behaviour when nodes leave in a restricted area. For those simulations we use a DST with 50 nodes in which we delete 30 nodes. The inferior and superior limits for the nodes size are respectively 2 and 4. First we delete node randomly from the DST. This result will be the witness. Then, we focus on a restricted area. We ran 20 simulations for both situations and note the total number of messages needed for all deletes. Both curves are shown on figure 6 (a). The average cost of random deleting is 1374 messages for 30 leaving nodes. When deleting in a restricted area the cost is 1555 messages for the same number of leaving nodes. The probability of merging nodes is higher when deleting from the same area. The merging operation has an important cost, that explain this result. Thus a good merging algorithm is paramount. We focus on the reorganisation cost in the next paragraph.

Reorganisation cost We use a DST with the smallest possible limits (2 and 4). We create a DST with 90 nodes and 5 levels. We delete 88 nodes so that we can study all the



(a) Deleting into different areas.



(b) Reorganisation cost.

Figure 6: Cost study of deleting.

steps of merges. The results are given on figure 6 (b). We observe peak for series 2 and 3 just before the tenth delete. This peak corresponds to the merging operation that implies the delete of the level 5. All peaks are due to reorganisation. The peaks are very pronounced because after a reorganisation the DST is well balanced.

Average cost of deleting. During our simulation we observe a linear increase of the number of messages in function of the level. Note that, in the studied case we initiate a very fragile DST with limits of 2 and 4. Thus, by studying the worst case we have the worst behaviour.

4. Description of the Simulations

The simulations presented in section 5 use two algorithms. The first is a TTL-based flooding algorithm that is used with the graph and tree topologies [4]. The second use the DST structure to run a traversal with similar properties. Fig. 7 illustrates the four steps of the execution of this algorithm on a DST of height 3. The request is initiated by the gray node on step 1. Then it broadcasts the request in the subtrees of level 1 on step 2, level 2 on step 3 and level 3 on step 4. Complete algorithms are given in [2].

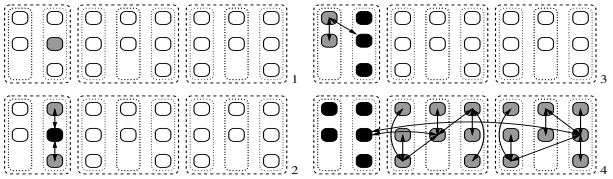


Figure 7: Example of a DST's traversal

Our first idea was to simulate and study the DST behavior on the Internet. However, experiences on Internet cannot be reproduced as the simulation conditions cannot be two times the same. Thus, we restrict to a model where all computers have one link, an access to every other computer through a central router and the bandwidth is limited. This model is not far from a realistic Internet model if we consider different bandwidth for different links and FIFO queues that assure that there is no more than one message in a link at any time. This way, we can mesure the impact of different factors on the execution performances.

Exact details of the simulations can be found by downloading the simulator [1]. We perform simulations for populations of 10, 100, 1 000 and 10 000 peers to study the behaviour of the algorithms when the overlay network scales up.

For all topologies the TTL is set to 10. Hundred different types of resources are available, and each computer has a probability of 10 % to own a resource for each type. Each search request stops either when it finds a node with the requested resource, when the TTL is reached or when the whole structure is traversed. However, we noticed that a maximum depth of four is generaly enough to find a resource whith this probability to find the resource.

About the overlay topologies characteristics, trees are bidirectional and their arity is 5. Graphs are also bidirectional, connected and the degree of each node is 5. Finally the DST is made in a way that each node has 5 children. These degrees were chosen because they show the best performances in our simulations. More precisely, we ran few tests at various scales to find these optimal degrees. Then, we used them for all our simulations by considering that these degrees are always optimal in our experiments. However, these values depend on the links throughput and the probability to find a service. Changing one of these parameters implies that the chosen degrees would not longer be optimal.

5. Discussion about the simulation results

In this section, we discuss the results obtained by the simulations. For each simulated overlay network, we compare the performances of the three tested topologies. The two performance criteria that we are interested about in this article are the average time taken to process a search and its variation depending on the average load of the system. The system load is defined by the requests arrival rate or frequency: the number of queries that are initiated per second for the whole network. In the last section, we explain how the three topologies scale up.

1 http://lifc.univ-fconte.fr/∼philippe/pub/simulator-12-06.tar.gz

5.1. 10 peers overlay networks

The simulations results for 10 peers overlay networks are displayed in Fig. 8. The simulations show that the average time needed to process a request depends on the requests arrival rate. This is an ordinary observation. When the number of initiated requests grows, the system becomes more and more loaded and messages spend more time in a waiting queue before being sent.

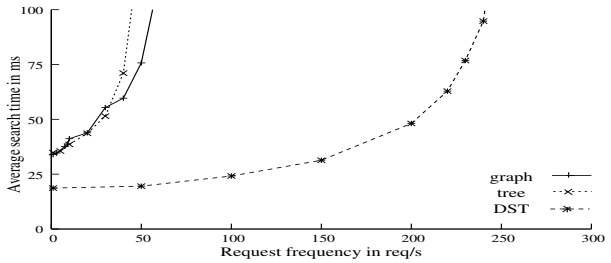


Figure 8: Performances for 10 peers networks

When the number of requests that enter the system becomes higher than the number of requests that leave it, the system becomes saturated. This saturation is easily identifiable for the DST on Fig. 8: the average time needed to process a request grows slowly for frequencies from 1 req.s^{-1} to a frequency of 150 req.s^{-1} ; but it grows very quickly for frequencies greater than 200 req.s^{-1}. On the other hand, the graph and the tree become very quickly saturated.

For a 10 peers overlay network, the simulations tell us that graphs and trees have similar performances. This is interesting because we expected to get better performances with the graph in term of time used to process a request when the system is not loaded (0.1 req.s^{-1}) as explained in section 5.2. The more plausible explanation of this result is that some requests completely traverse the graph as no resource is found (we only have 10% of chance to find a resource on each peer and we only have 10 peers). So, these requests need an additional round to check that no other peers can be contacted. This is the final round where lots of messages are sent to traverse both ways the latest links, not marked as flooded, although no untraversed peer remains. Due to this additional round, the traversal depends on the number of links rather than on the number of nodes and affects the global average search time.

Studies of individual load of peers also explain why the tree and the graph become saturated on a similar way: bottlenecks of both topologies have to support similar load. This can be explained by the fact that the tree has few bottlenecks that send an average of, say $\bar{x}$ messages. With the graph traversal every peer also sends an average of $\bar{x}$ messages, as the number of messages depends on the number of links. Because every peer has the same bandwidth, and because the topologies performances are limited by their bot-

tlenecks, both topologies become overloaded in a similar way at this scale. The DST has the best behaviour for these simulations. Because a DST is based on trees, a search request only needs $2.n$ messages to query n peers, which is less than the graph. But because it distributes the load of father nodes between its children, it does not suffer from the tree bottlenecks. Thus, the DST can support much more load than the other topologies at a scale of 10 peers. Note that the DST also generates a lower search time than the tree when the system is not loaded.

5.2. 100 peers overlay networks

Fig. 9 presents the simulations results for 100 peers. Like before, the three topologies saturate when the query arrival rate becomes too high. From the simulations, it is clear that the tree has the worst performances in term of supported load. 100 req.s^{-1} is enough to overload trees while graphs and DST start to overload for a frequency of 700 req.s^{-1}. Before reaching this point, the average search time grows slightly with the average load of the system.

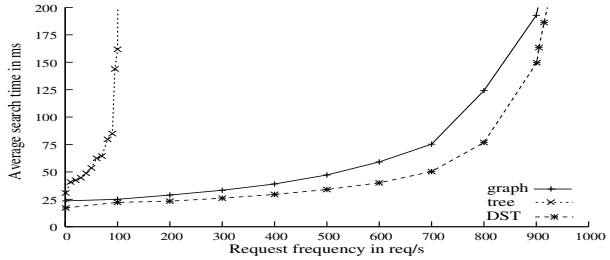


Figure 9: Performances for 100 peers networks

Graphs performances are roughly similar to the DST ones in term of supported load and average search time. Because each search needs to query a small part of the graph, few links are traversed to access already contacted peers. Thus, few peers are contacted twice by the same request and the number of exchanged messages is, in a wide approximation, roughly similar to the tree's one. It depends on the number of nodes rather than on the number of links. For this reason, as graphs inherently distributes their load between their peers and as a DST is a tree that distributes the father's nodes load between its descendants, the performances of both graphs and DST are approximately similar in term of average query time and supported load.

We observed that tree topologies have a higher average search time than graphs even when the average load is very low. This is counter-intuitive if we think that the average query time only depends on the number of exchanged messages and that the tree based traversal generates an optimal number of messages. In a balanced tree with an arity of 5, there are lot more leaves than non-leaf nodes. Because every peer has the same probability to initiate a search, the majority of searches are initiated by leaf nodes. These searches are not efficient because the first step of a search only queries one peer, its father, when the same step queries 5 peers in a graph or in a DST. At the second step the tree queries 5 peers when the graph or the DST can query 25 peers. Further, the tree behaves as if it always has one round late compared to the two other topologies, because tree leaves have one more hop to cross when they issue a search request. Being one round behind the other topologies is translated by a higher latency which explains why the average search time of the tree is more important than the two others even when the system is under light load.

Same simulation were made for 1 000 and 10 000 peers overlay networks. Results shown the same type of curves. For 1 000 peers, a load of 300 req.s^{-1} is enough to saturate the tree and it starts to overload around 1 000 req.s^{-1} for 10 000. Graphs and DST start to overload around 8 000 req.s^{-1}with 1 000 peers and support at least a load of 40 000 req.s^{-1}with 10 000. Before overloading, the average search time of DSTs and graphs grows slowly when the load increases and it linearly depends on the load.

5.3. Scalability of the three topologies

Until now, we have discussed the performances for each scale separately. Here we present how the number of peers affects the performances of each topologies. For Fig. 8 and Fig. 9, we note that adding new peers always allows to increase the performances in term of supported load. But, depending on the topology, the supported load per node may decrease. For the tree topology, 40 req.s^{-1} saturate a tree of 10 peers, 100 req.s^{-1} saturate a tree of 100 peers, 300 req.s^{-1} saturate a tree of 1 000 peers and 1 000 req.s^{-1} saturate a tree of 10 000 peers. This is not very efficient because multiplying the number of peers by 100 (from 100 peers to 10 000 peers) only multiplies the supported load by a factor 10 (from 100 req.s^{-1} to 1 000 req.s^{-1}). We can conclude that the tree topology does not scale and the reason is clearly the bottlenecks generated by non-leaf nodes. The DST has a good scalability. The supported load grows linearly with the number of peers: 150 req.s^{-1} for 10 peers, 700 req.s^{-1} for 100 peers, 8 000 req.s^{-1} for 1 000 peers and more than 40 000 req.s^{-1} for 10 000 peers. When the system is not saturated, the performances, in term of average search time, are stable: it varies from 25 ms to 75 ms whatever the number of peers is. A performance of 150 req.s^{-1} on 10 peers for a DST seems to be inconsistent with the performance of 700 req.s^{-1} for 100 peers, but this is normal. There is only 10 peers, so two steps searches only contact 10 peers when the same search contact 25 peers in a 100 peers DST. Less peers contacted means less messages generated and implies that the overlay networks supports mode load. The scalability of the graph is similar to the DST's one except for small

scale where it is worst. This similarity comes from the fact that both topologies behave in a similar way when the probability to find a resource on each peer is fixed (10 % in our simulation) and that the number of peers is high enough. If the scale is too small, then queries contact several times the same peers and graphs perform badly.

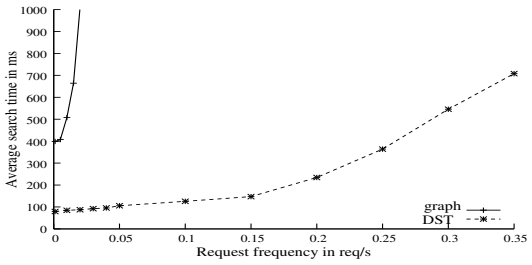


Figure 10: Performances for 1 000 peers networks when no resources are available

However, if the probability to find a resource is low enough that peers receive a query several time, then graphs' performances cannot cope with the DSTs' ones. Fig. 10 presents the performances of graphs and DSTs of 1 000 computers when the probability to find a resource is 0 %. In this case, the DST's performances are better than the graph's ones because a DST sends fewer messages and distributes fairly its load between computers. Note that a frequency of 1 request every 5 seconds is enough to saturate a DST of 1 000 computers. This is normal as TTL-based flooding algorithms, which generate several waves of search, are not efficient for this kind of applications. A directory, like a DHT, should be used instead.

Other simulations on influence of service population and load balancing were run in [2].

6. Conclusion

If the performances of search algorithms only depend on the number of messages generated by a search request then, in theory, these algorithms are more efficient on a tree than on a graph. But, taking into account bottlenecks, graphs are more resistant than trees.

The Distributed Spanning Tree (DST) is a tree based structure that connects a set of computers while distributing the load of parent nodes between their children. This structure is in theory better suited to run flooding based search algorithms than trees and graphs topologies. It is designed to send an optimal number of messages like a tree without suffering from topological bottlenecks. The simulations presented in the paper confirm the interest of this approach and the DST always has better performances in term of average search time and supported load. Moreover, we found that, thanks to the DST structure, the implementation of traversal algorithms is simpler for the DST than for graphs.

As these results are encouraging, we are currently working on the dynamics of the structure. We already designed the algorithms to dynamically construct and manage a DST. Simulations are carried out to observe the evolution and robustness of the DST in a context of short lived peers. We will focus on that point for further works.

References

[1] S. Dahan. Distributed Spanning Tree algorithms for large scale traversals. In *Procedings of the11th International Conference on Parallel and Distributed Systems, ICPADS 2005*, volume 1, pages 453–459, Fukuoka, Japan, July 2005. IEEE Computer Society.

[2] S. Dahan, A. Dobrila, J. Nicod, and L. Philippe. Performances study of management and traversal algorithms on the DST overlay network. Research Report RR2007-02, LIFC, Nov. 2007.

[3] S. Dahan, J. Nicod, and L. Philippe. The Distributed Spanning Tree: A scalable interconnection topology for efficient and equitable traversal. In *Proceedings of CCGrid 2005 Workshop Global and Peer-2-Peer Computing*, Cardiff, UK, May 2005. IEEE Press. CD-ROM.

[4] S. Dahan, J.-M. Nicod, and L. Philippe. Scalability in a GRID server discovery mechanism. In *10th IEEE Int. Workshop on Future Trends of Distributed Computing Systems, FTDCS 2004*, pages 46–51, Suzhou, China, May 2004. IEEE Press.

[5] G. Fletcher and S. H. Unstructured peer-to-peer networks: Topological properties and search performance. In *3rd Int Workshop on Agents and Peer-to-Peer Computing (AP2PC) at AAMAS 2004*, pages 14–27, New York, 2004.

[6] R. Gaeta, G. Balbo, S. Bruell, M. Gribaudo, and M. Sereno. A simple analytical framezork to analyze search strategies in large-scale peer-to-peer networks. *Performance Evaluation*, 62(1–4):1–16, Oct. 2005.

[7] D. E. Knuth. *The Art of Computer Programming*, volume 3, chapter 6.2.4. Addison-Wesley, 75 Arlington Street, Suite 300, Boston, MA 02116, 1998.

[8] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lims. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 7(2), April 2005.

[9] R. E. McGrath. Discovery and its discontents: Discovery protocols for ubiquitous computing. Technical Report UIUCDCS-R-99-2132, Department of Computer Science University of Illinois, Champaign, IL, USA, April 2000.

[10] L. Qin, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ACM SIGMETRICS 2002*, pages 258–259, Marina Del Rey, California, 2002.