A Comparison of Peer-to-Peer Search Methods

Dimitrios Tsoumakos
Computer Science Department
University of Maryland
dtsouma@cs.umd.edu

Nick Roussopoulos
Computer Science Department
University of Maryland
nick@cs.umd.edu

ABSTRACT

Peer-to-Peer networks have become a major research topic over the last few years. Object location is a major part in the operation of these distributed systems. In this work, we present an overview of several search methods for *unstructured* peer-to-peer networks. Popular file-sharing applications, through which enormous amounts of data are daily exchanged, operate on such networks. We analyze the performance of the algorithms relative to their success rates, bandwidth consumption and adaptation to changing topologies. Simulation results are used to empirically evaluate their behavior in direct comparison.

1. INTRODUCTION

Peer-to-Peer (hence P2P) computing represents the notion of sharing resources available at the edges of the Internet. After its initial success, which resulted in the subsequent appearance of numerous P2P systems, it now emerges as the dominant model for the networks of the future. The P2P paradigm dictates a fully-distributed, cooperative network design, where nodes collectively form a system without any supervision. Its advantages (although application-dependent in many cases) include robustness in failures, extensive resource-sharing, self-organization, load balancing, data persistence, anonymity, etc.

Today, the most popular P2P applications operate on *unstructured* networks. In these networks, peers connect in an ad-hoc fashion, the location of the documents is not controlled by the system and no guarantees for the success of a search are offered to the users. Bandwidth consumption attributed to such applications amounts to a considerable fraction (up to 60%) of the total Internet traffic [1]. It becomes obvious that the nature of data discovery and retrieval is of great importance to the user and the broad Internet community.

In this work, we examine the problem of object discovery in *unstructured* P2P networks. Nodes make requests for objects they want to retrieve and cannot be found in their local repositories. The search process includes aspects such as the query-forwarding method, the set of nodes that receive query-related messages, the form of these messages, local processing, locally stored indices and their maintenance, etc.

Current search algorithms aim for bandwidth-efficient and adaptive object discovery for these networks. Search methods can be categorized as either *blind* or *informed*. In a *blind* search, nodes hold no information that relates to document locations, while in *informed* methods, there exists a centralized or distributed directory service that assists in the search for the requested objects.

In this paper, we describe current approaches from both categories and analyze their performance. We focus on the behavior of these algorithms for each of the following metrics: Efficiency in object discovery, bandwidth consumption and adaptation to rapidly changing topologies. The first metric measures search *accuracy* and the number of discovered objects per request. The latter is important for many applications, as it gives users a much broader choice for object retrieval. Minimizing message production always represents a high-priority goal for all distributed systems. Finally, it is important that any search algorithm adapts to dynamic environments, since in most P2P networks users frequently enter and leave the system, as well as update their collections.

To evaluate our analysis, we simulate many of the described methods and present a quantitative comparison of their performance. We also identify the conditions under which each method would be most or least effective.

In Section 2 we present related work. Section 3 categorizes and describes current search techniques, while in Section 4 we present the simulation results.

2. RELATED WORK

Peer-to-Peer networks have been studied a lot in the last few years. A large amount of information for P2P computing with taxonomies, definitions, current trends, applications and related companies can be obtained at [2, 3], as well as individual sources (e.g., [4, 5]). P2P computing is also described in [6], with basic terminology, taxonomies and description of some systems. A brief summarization of Gnutella [7] and Napster [8] search, together with ap-

Copyright is held by the author/owner.
International Workshop on the Web and Databases (WebDB).
June 12-13, 2003, San Diego, California.

proaches for *structured* networks are also included.

Gnutella and Napster are the focus of two measurement studies; [9] attempts a detailed characterization of the participating end-hosts, while [10] measures the locality of stored and transferred documents. In [11], a traffic measurement for three popular P2P networks is being conducted at the border routers of a large ISP. Extensive results for traffic attributed to HTTP, Akamai and P2P systems are also presented in [12].

Quantitative comparisons between the search methods in [13, 14] and the original Gnutella algorithm are presented in these two papers. Their main comparison metric is bandwidth consumption. The work in [15] presents a thorough comparison between the proposed algorithm and two blind search schemes ([14, 16]) on a variety of metrics.

Our work focuses exclusively on search in *unstructured* P2P networks, together with an experimental comparison of the methods under certain criteria.

3. P2P SEARCH ALGORITHMS

3.1 Our Framework

In our general framework, peers communicate either when they search for an object or when they share one. In this work, we examine proposed algorithms for the first part only.

Each peer retains a local collection of documents, while it makes requests for those it wishes to obtain. The documents are stored at various nodes across the network. Peers and documents (or objects) are assumed to have unique identifiers, with object IDs used to specify the query target. Search algorithms cannot in any way dictate object placement and replication in the system. They are also not allowed to alter the topology of the P2P overlay. Nodes that are directly linked in the overlay are *neighbors*. A node is always aware of the existence and identity of its neighbors. Nodes can also keep *soft state* (i.e., information that is erased after a short amount of time) for each query they process. Each search is assigned an identifier, which, together with the soft state, enables peers to make the distinction between new queries and duplicate ones received due to a cycle.

A search is *successful* if it discovers at least one replica of the requested object. The ratio of successful to total searches made is called the *success rate* (or *accuracy*). A search can result to multiple discoveries (or *hits*), which are replicas of the same object stored at distinct nodes. A global TTL parameter represents the maximum hop-distance a query can reach before it gets discarded.

3.2 Search Taxonomy

There are two possible strategies for search in an unstructured P2P network: Search in a *blind* fashion, trying to propagate the query to a sufficient number of nodes in order to satisfy the request; or utilize information about document locations and thus perform an *informed* search. The semantics of the used information range from simple forwarding hints to exact object locations. The placement of this information can also vary: In centralized approaches (e.g., [8]), a central directory known to all peers exists. In distributed approaches ([17, 13], etc.), each individual peer holds a piece of the information.

One can also categorize search algorithms according to the model of P2P network they are designed to operate on. In *pure* P2P systems, all participating peers play both the roles of the client and the server. Other algorithms operate on *hybrid* P2P architectures, where certain nodes assume the role of a *super-peer* and the rest become *leaf-nodes*. Each super-peer acts as a proxy for all its neighboring leaves by indexing all their documents and servicing their requests.

3.3 *Blind* Search Methods

GNUTELLA [7]: The original Gnutella algorithm uses flooding (BFS traversal of the underlying graph) for object discovery and contacts all accessible nodes within the TTL value. Although it is simple and manages to discover the maximum number of objects in that region, the approach does not scale, producing huge overhead to large numbers of peers.

Modified-BFS [13]: This is a variation of the flooding scheme, with peers randomly choosing only a ratio of their neighbors to forward the query to. This algorithm certainly reduces the average message production compared to the previous method, but it still contacts a large number of peers.

Iterative Deepening: Two similar approaches that use consecutive BFS searches at increasing depths are described in [18, 14]. These algorithms achieve best results when the search termination condition relates to a user-defined number of hits and it is possible that a "small" flood will satisfy the query. In a different case, they produce even bigger loads than the standard flooding mechanism.

Random Walks [14]: In *Random Walks*, the requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These queries are also known as *walkers*. A walker terminates either with a success or a failure. Failure can be determined by two different methods: The TTL-based method and the *checking* method, where walkers periodically contact the query source asking whether the termination conditions have been satisfied.
The algorithm's most important advantage is the significant message reduction it achieves. It produces $k \times TTL$ messages in the worst case, a number which seldom depends on the underlying network. Simulation results in [14, 15] show that messages are reduced by more than an order of magnitude compared to the standard flooding scheme. It also achieves some kind of local "load balancing", since no nodes are favored in the forwarding process over others.
The most serious disadvantage of this algorithm is its highly variable performance. Success rates and number of hits vary greatly depending on network topology and the random choices made. Finding an object depends slightly on its hop-distance. Another drawback of this method is its inability to adapt to different query loads. Queries for popular and unpopular objects are treated in the exact same manner. *Random Walkers* cannot *learn* anything from its previous successes or failures, displaying high variability in

all ranges of requests.

Recently, two new search protocols which operate on *hybrid* P2P networks made their appearance:
GUESS [16]: This algorithm builds upon the notion of *Ultrapeers* [19]. Each ultrapeer is connected to other ultrapeers and to a set of leaf-nodes (peers shielded from other nodes), acting as their proxy. A search is conducted by iteratively contacting different ultrapeers (not necessarily neighboring ones) and having them ask all their leaf-nodes, until a number of objects are found. The order with which ultrapeers are chosen is not specified.
Gnutella2 [20]: In Gnutella2, when a super-peer (or *hub*) receives a query from a leaf, it forwards it to its relevant leaves and also to its neighboring hubs. These hubs process the query locally and forward it to their relevant leaves. No other nodes are visited with this algorithm. Neighboring hubs regularly exchange local repository tables to filter out unnecessary traffic between them.
Although the details of these protocols are still formulating, we observe they rely on a dynamic hierarchical structure of the network. They present similar solutions for reducing the effects of flooding by utilizing the structure of hybrid networks. The number of leaf-nodes per super-peer must be kept high, even after node arrivals/departures. This is the most important condition in order to reduce message forwarding and increase the number of discovered objects.

3.4 *Informed* Search Methods

Intelligent-BFS [13]: This is an *informed* version of the *modified-BFS* algorithm. Nodes store query-neighborID tuples for recently answered requests from (or through) their neighbors in order to rank them. First, a peer identifies all queries similar to the current one, according to a query similarity metric; it then choses to forward to a set number of its neighbors that have returned the most results for these queries. If a hit occurs, the query takes the reverse path to the requester and updates local indices.
This approach focuses more on object discovery than message reduction. At the cost of an increased message production compared to *modified-BFS* (because of the update process), the algorithm increases the number of hits. It achieves very high accuracy, enables knowledge sharing and induces no overhead during node arrivals/departures. On the other hand, its message production is very large and only increases with time as knowledge is spread over the nodes. It shows no easy adaptation to object deletions or peer departures. This happens because the algorithm does not utilize negative feedback from searches and forwarding is based on ranking. Finally, its accuracy depends highly on the assumption that nodes specialize in certain documents.

APS [15]: In *APS*, each node keeps a local index consisting of one entry for each object it has requested per neighbor. The value of this entry reflects the relative probability of this node's neighbor to be chosen as the next hop in a future request for the specific object. Searching is based on the deployment of k independent walkers and probabilistic forwarding. Each intermediate node forwards the query to one of its neighbors with probability given by its local index. Index values are updated using feedback from the walkers. If a walker succeeds (fails), the relative probabilities of the nodes on the walker's path are increased (decreased). The update procedure takes the reverse path back to the requester and can take place either after a walker miss (*optimistic* update approach), or after a hit (*pessimistic* update approach).
APS exhibits many plausible characteristics as a result of its *learning* feature. Every node on the deployed walkers updates its indices according to search results, so peers eventually share, refine and adjust their search knowledge with time. Walkers are directed towards objects or redirected if a miss or an object deletion occurs. *APS* is also very bandwidth-efficient, achieving very similar levels with *Random Walks*. It induces zero overhead over the network at join/leave/update operations and displays a high degree of robustness in topology changes. The *s-APS* modification adaptively switches between the optimistic and pessimistic approaches and further reduces the amount of messages. Finally, [15] showed that the algorithm's majority of discovered objects are located close to the requesters. These advantages are mainly seen when many different peers contribute with big workloads. This is because *APS* gains from knowledge build-up and increased peer cooperation.

Local Indices [18]: Each node indexes the files stored at all nodes within a certain radius r and can answer queries on behalf of all of them. A search is performed in a BFS-like manner, but only nodes accessible from the requester at certain depths process the query. To minimize the overhead, the hop-distance between two consecutive depths must be $2r + 1$.
This approach resembles the two search schemes for hybrid networks. The method's accuracy and hits are very high, since each contacted node indexes many peers. On the other hand, message production is comparable to the flooding scheme, although the processing time is much smaller because not every node processes the query. The scheme also requires a flood with $TTL = r$ whenever a node joins/leaves the network or updates its local repository, so the overhead becomes even larger for dynamic environments.

Routing Indices (RI) [17]: Documents are assumed to fall into a number of thematic categories. Each node knows an approximate number of documents from every category that can be retrieved through each outgoing link (i.e., not only from that neighbor but from all nodes accessible from it). The query termination condition always relates to a minimum number of hits. The forwarding process is similar to DFS: A node that cannot satisfy the query stop condition with its local repository will forward it to the neighbor with the highest "goodness" value. Three different functions which rank the out-links according to the expected number of documents discovered through them are also defined. The algorithm backtracks if more results are needed.
This is another keyword-search approach which trades index maintenance overhead for increased accuracy. While a search is very bandwidth-efficient, RIs require flooding in order to be created and updated, so the method is not suitable for highly dynamic networks. Moreover, stored indices can be inaccurate due to thematic correlations, over-counts or under-counts in document partitioning and network cycles.

In [21], each node holds d *bloom* filters for each neighbor. A filter at depth i summarizes documents that can be found i hops away through that specific link. Nodes forward queries to the neighbor whose smaller depth bloom filter matches a

hashed representation of the object ID. After a certain number of steps, if the search is unsuccessful, it is handled by a deterministic algorithm instead of backtracking.
This method exhibits the same characteristics as the two previous ones. The scheme's expectation is to find only one replica of the object with high probability. Index maintenance requires flooding messages initiated from nodes that arrive or update their collections.

Distributed Resource Location Protocol (DRLP) [22]: Nodes with no information about the location of a file forward the query to each of their neighbors with a certain probability. If an object is found, the query takes the reverse path to the requester, storing the document location at those nodes. In subsequent requests, nodes with indexed location information directly contact the specific node. If that node does not currently obtain the document, it just initiates a new search as described before.
This algorithm initially spends many messages to find the locations of an object. In subsequent requests, it might take only one message to discover it. Obviously, a small message production is achieved only with a large workload that enables the initial cost to be amortized over many searches. In rapidly changing networks, this approach fails and more nodes have to perform blind search. This also affects the number of hits: If many blind searches are made, then many results are found; if many direct queries take place, then only one replica is retrieved. So, this scheme is very dependent on network/application parameters.

4. SIMULATION RESULTS

In this section we present results for six of the described methods (*GUESS, Random Walks, Modified-BFS, Intelligent-BFS, s-APS, DRLP*). Three of the simulated methods are representative *blind* search schemes. The rest are *informed* methods that do not require user-initiated index updates.

Due to space limitations, we will briefly summarize our simulation model here. More details can be found in [15]. We use a *random* graph of 10000 nodes and an average degree of 10 generated by GT-ITM [23] to simulate our P2P overlay structure. We assume a *pure* P2P model, where all peers equally make and forward requests. Results comparing *APS* and *GUESS* in *hybrid* topologies can be found in [15].

Queries are made for 100 objects, with object 1 being the most popular and object 100 the least. Qualitatively similar results are produced when using a larger number of objects in the simulations. Objects are stored over the network according to the *replication distribution*, while nodes make requests according to the *query distribution* (e.g., popular objects get many more requests than unpopular ones). A zipfian distribution with parameter $a = 0.82$ is used to model both and achieve workloads similar to the observations in [10]: The highest-ranked 10% of objects amount to about 30% of the total number of stored objects and receive about 30% of all requests. Requester nodes are randomly chosen and represent about 20% of the total number of nodes. Each requester makes about 1500 queries. The TTL parameter was set to 5 for all algorithms, since larger values produced very similar results.

To simulate dynamic network behavior, we insert "on-line" nodes and remove active ones with varying frequency. We always keep approximately 80% of the network nodes active, while arriving nodes start functioning without any prior knowledge. The objects are also re-distributed (less frequently though) to model file insertions and deletions. Object re-location always follows the initial distribution parameters.

The *Intelligent-BFS* method was modified to allow for object-ID requests. Index values at peers now represent the number of replies for an object through each neighbor. Nodes simply choose the 5 highest ranked neighbors in query forwarding. For *Modified-BFS* and *DRLP*'s blind search, nodes randomly choose half of their neighbors (5 on average) to forward a query to. In our *GUESS* implementation, peers deploy k random walkers with $TTL = 4$. The last nodes on the paths of these walkers forward the query to all their neighbors. In our simulations, $k = 12$ for *Random Walks, APS* and *GUESS*.

We simulate the algorithms at three different environments: In the static case there are no dynamic operations. In the less dynamic setting, the topology changes on average 240 times and objects are relocated 120 times at each run. In the highly dynamic setting, the topology changes about 1200 times and objects are relocated about 500 times during each run.

Figures 1, 2 and 3 present the results for the six methods. As expected, both *Modified* and *Intelligent-BFS* show extremely high accuracy and return many hits. Although their message production is an order of magnitude less than that of the original Gnutella scheme, they still produce almost 2 orders of magnitude more than the other four schemes. The informed method produces 3 times more messages, but also manages to find 3 times more objects. Both algorithms are not affected by the changing environment and achieve similar results in all settings. For environments similar to our setup, the modified method will be preferred to the intelligent one, since its performance is equally high and it is much simpler. We expect that the informed method will perform better in specialized environments (like the one described in [13]), mainly in the number of hits, which is one of the algorithm's goals.

Random Walks displays low accuracy and finds less than one object per query on average. Its bandwidth consumption is quite low (between 33 and 40 messages) and its overall performance is hardly affected by the dynamic operations. *GUESS* behaves similarly, with the exception of being steadily over 60% accurate and discovering about 2 objects per query. On the other hand, it produces twice the number of messages of *Random Walks*. In general, these algorithms appear very robust to increased network variability. This is reasonable, as walkers are randomly directed with no regard to topology or previous results.

s-APS achieves a success rate of over 90% in the static run, a number that drops by 6% and 15% in the following settings. The metric that is reasonably affected is the number of discovered objects, which are almost cut to a third (from 6 to 2.2). This happens because it takes some time for the learning feature to adapt to the new topology and

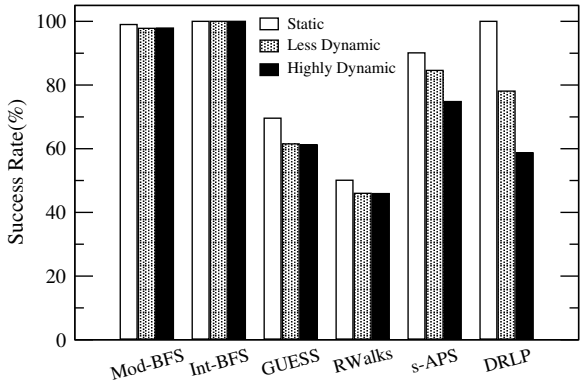

Figure 1: Success rate

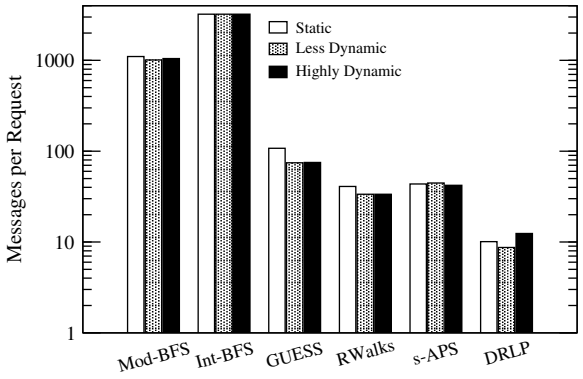

Figure 2: Messages per query

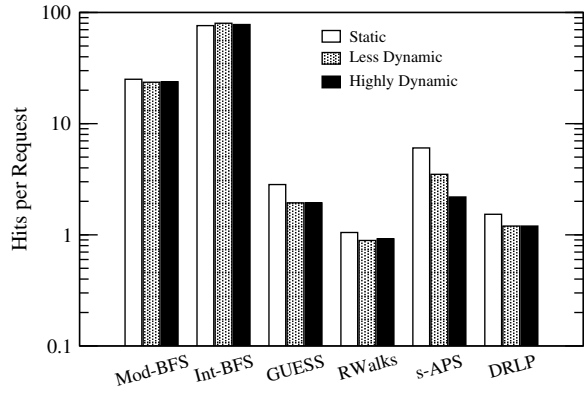

Figure 3: Hits per query

paths to discovered objects frequently "disappear". On the other hand, it manages to keep its messages at the same levels, producing about 42 per search. *s-APS* exhibits a very good overall performance compared to the four non-BFS related schemes, being much more bandwidth-efficient than the BFS-related techniques.

The *DRLP* algorithm exhibits some interesting characteristics. First, its message production is very low (from 9 to about 12 messages per request), with only a small increase as the network becomes more dynamic. Our simulations count the direct contact of a node in *DRLP* as one message, although a link between them might not exist in the overlay. Dynamic behavior causes the stored addresses to become more frequently "stale", thus the initial flooding is performed more often. This is the reason for the decrease in its accuracy from 100% in the static case to 80% and 60% respectively. *DRLP* produces the same amount of messages for its initial search with *Modified-BFS*, so it needs many successful requests to amortize this initial cost. The number of objects it discovers is small, ranging from 1.5 to 1.2. If *DRLP* is forced to use flooding many times, then the number of hits increases. If it is successful and produces few messages, then it only finds one replica per request. Despite this fact, we notice that it proves very bandwidth-efficient, while one would expect an increase to justify more flooding requests. This is due to the fact that, with many nodes making requests, most of them obtain a pointer for every object after a while. So, even if some node initiates a flood, most of its neighbors will only forward to one node. This scheme seems ideal for relatively static environments and large workloads, with the exception that the number of hits will be very close to one. Another observation we made is that *DRLP* is affected more by object relocation than by node departures.

Figure 4 shows how object popularity affects the accuracy of the six schemes in the highly dynamic environment. The results are similar for the other two settings. Popular objects are stored in more nodes and receive more requests. Popularity decreases as we move to the right along the x-axis. The first data point represents the accuracy of the methods for objects 1-10, the second for objects 11-20, etc. The two BFS methods exhibit high accuracy with *Intelligent-BFS* performing marginally (2-5%) better than *Modified-BFS*. *Random Walks, GUESS* and *s-APS* show decreasing accuracy as popularity drops, with *s-APS* clearly outperforming the other two. This difference becomes large for medium-popular objects. *DRLP* exhibits increasing accuracy as the popularity drops. This can be explained by the fact that less popular objects receive considerably fewer queries. Therefore, object relocations and node departures which affect the algorithm happen less frequently during requests for such objects.

Figure 5 shows how the number of hits is affected by the number of requests per object. Each object is uniformly stored in about 2% of the network nodes. The two BFS schemes show a stable performance as in the previous simulations, with the exception that *Modified-BFS* now produces more messages and discovers a few more objects than *Intelligent-BFS* (3900 to 3500 messages, 70 to 65 hits respectively). We notice that *GUESS* and *Random Walks* do not gain from the increased workload and exhibit results similar to the previous simulations. *s-APS* takes advantage of the increased requests to discover almost 7 times more objects. On the other hand, *DRLP* discovers a decreasing number of documents (nearly 7 times fewer hits at the final run), as the requests for each object increase. This happens because with more requests, fewer nodes perform flooding and only one object is discovered in almost every search. At the same time, the message production of *DRLP* also drops for the same reason.

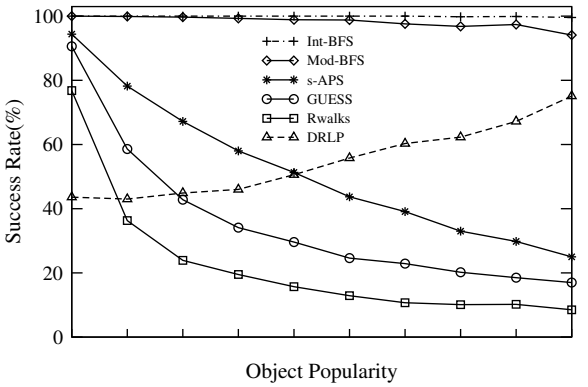

Figure 4: Success rate vs object popularity. Popularity decreases from left to right

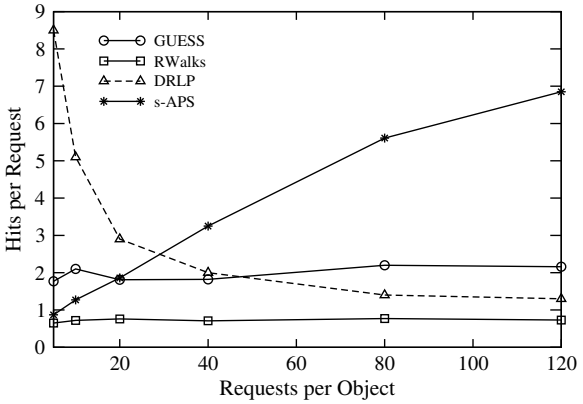

Figure 5: Number of hits as the number of requests per object increases in a static network

5. CONCLUSIONS

This paper presents an overview of current search techniques for *unstructured* P2P networks. Our analysis and simulations focus on three metrics: accuracy, bandwidth production and discovered objects. Flood-based schemes (e.g. *Int-BFS, Mod-BFS*) exhibit high performance at a very high cost. Other *blind* methods (e.g. *Random Walks, GUESS*) are simple and can greatly reduce bandwidth production but generally fail to adapt to different workloads and environments. Conversely, most *informed* methods achieve great results but incur large overheads due to index updates. *DRLP* and *s-APS* require no costly updates. The former performs best in relatively static environments, while the latter uses its adaptive nature to achieve good performance at low cost. *s-APS* particularly favors nodes with a prolonged stay in the network and the discovery of popular objects.

For future work, we plan on giving emphasis to the index update process and the imposed overhead over the network. This will allow for a more thorough comparison of the informed search methods.

6. REFERENCES

[1] The impact of file sharing on service provider networks. An Industry White Paper, Sandvine Inc.

[2] openP2P website: http://www.openp2p.com.

[3] Peer-to-peer working group: http://www.peer-to-peerwg.org/.

[4] Project JXTA: http://www.jxta.org.

[5] Microsoft .NET: http://www.microsoft.com/net.

[6] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP, 2002.

[7] Gnutella website: http://gnutella.wego.com.

[8] Napster website: http://www.napster.com.

[9] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, Un. of Washington, 2001.

[10] J. Chu, K. Labonte, and B. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *SPIE*, 2002.

[11] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *SIGCOMM Internet Measurement Workshop*, 2002.

[12] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *OSDI*, 2002.

[13] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *CIKM*, 2002.

[14] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS*, 2002.

[15] D. Tsoumakos and N. Roussopoulos. Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks. Technical Report CS-TR-4451, Un. of Maryland, 2003.

[16] S. Daswani and A. Fisk. Gnutella UDP Extension for Scalable Searches (GUESS) v0.1.

[17] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *ICDCS*, July 2002.

[18] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *ICDCS*, 2002.

[19] A. Singla and C. Rohrs. Ultrapeers: Another Step Towards Gnutella Scalability.

[20] M. Stokes. Gnutella2 Specifications Part One: http://www.gnutella2.com/gnutella2_search.htm.

[21] S. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM*, 2002.

[22] D. Menascé and L. Kanchanapalli. Probabilistic Scalable P2P Resource Location Services. *SIGMETRICS Perf. Eval. Review*, 2002.

[23] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Infocom*, 1996.