

Adaptive Data Propagation in Peer-to-Peer Systems *

Thomas Repantis
Department of Computer Science & Engineering
University of California, Riverside
trep@cs.ucr.edu

March 11, 2004

1 Introduction

Peer-to-peer (P2P) systems have attracted a lot of interest, as a highly dynamic platform that enables autonomous computing nodes to share resources and services. Operation of all the peers as both clients and servers, and without a central coordinator eliminates possible bottlenecks in terms of scalability or reliability. Structured peer-to-peer systems, where objects (or their indices) are located at specific nodes, provide efficient location of objects, but have to maintain a lot of state information, while the nodes have limited autonomy. On the other hand, in unstructured peer-to-peer systems, where objects are located at random nodes, nodes can join the system quickly and even leave it unexpectedly. More advantages of unstructured peer-to-peer systems include their ability for self-organization, for adaptation to different loads, and for resiliency to node failures.

However, in an unstructured topology several design issues arise, one of the most challenging of them being the efficient search and retrieval of data. Specifically, we are looking for *a way to efficiently locate an object in a fully decentralized, self-organizing network, when a reference to that object is given*. An efficient search mechanism is important for the scalability of the system.

Traditionally, search in unstructured peer-to-peer networks is performed by flooding the immediate neighbors and propagating the search query hop-by-

hop, without taking into account the probability of a node to be able to provide the asked object. Hence, the search messages travel a large number of hops, wasting processing power of many nodes, and producing large amounts of network traffic, while the answer to the query is delayed.

Organizing the nodes according to their interests [8], as well as using Bloom filters to summarize the data that is stored in nodes [7] are two techniques that have been proposed to tackle the aforementioned problem.

When taking into account the interests, each peer tries to move closer in the network to a peer that frequently provides him with good results, by creating a direct connection with that peer.

By having access to a synopsis of the data stored in other peers, each node can decide where to propagate a query that cannot be answered locally, in order to maximize the probability for a fast reply. Hence, apart from having Bloom filters that represent the objects stored locally, each peer also shall maintain Bloom filters of objects stored in remote peers.

Bloom filters are compact data structures for probabilistic representation of a set, in order to support membership queries [1]. The downside of the use of Bloom filters, is that there is a small probability of false positives in membership queries; in other words an element might be incorrectly reported to be part of the set.

Since having synopses of the contents of all the peers is impossible due to bandwidth and storage limitations, the question is which peers' synopses to

*Course Project Report for CS253 - Distributed Systems, Spring 2004.

store, or to which peers to propagate our synopses. The answer is not as obvious as in a hierarchical organization, where the data of all the following nodes in the tree are summarized in the tree's root node [6]. Among the choices are:

- The immediate peers,
- the peers that have served us/replied to our requests, or the peers that have been served by us/got a reply to their requests by us.

Obviously, the last choice identifies the peers that share the same interests as we do, which may become our immediate peers in the future.

In order to adapt our decision to the current status of the network –and hence actively propagate synopses more efficiently– we need to take into account several parameters, which might include the following:

Parameters regarding other peers (the environment):

- Coherency requirements they might have for data they receive.
- Their stability, in terms of uptime, and history (past encounters).
- The number of connections they maintain, and the number of messages they route per time unit.
- Their proximity at the underlying physical network.
- The administrative domain they belong.
- Their general and current networking, processing, and storage capabilities.

Parameters regarding the local peer (ourselves):

- The number of queries we have received by a peer, and their frequency.
- The number of replies we have provided a peer with, and their frequency.
- The number of our stored objects.

- The popularity of our stored objects, in terms of queries received for those objects, possibly during a certain time period (e.g. between propagating new synopses).
- Our general and current networking, processing, and storage capabilities.

The update of the synopses is also an issue. Updates can be generated periodically, or whenever an object is added, deleted, or changed. Moreover whenever a peer joins the system updates need to be generated. Apart from the frequency of the updates, the recipients of them have also to be selected, according to the peers that keep a synopsis for the changing peer.

Our contribution lies in the fact that, *by propagating content synopses to peers that are selected adaptively, we make the search and retrieval of data in unstructured peer-to-peer networks faster, and more efficient in terms of bandwidth and processing power usage.*

2 System Model

We assume a logical network of peers that store objects. Each peer maintains connections with other peers. The network is decentralized and self-organizing, meaning that peers make their own decisions on which peers to connect to or to query for objects. Each peer builds a synopsis of its content and sends it to some of its neighbors. The recipients of the synopsis are selected adaptively, by taking into account the parameters mentioned previously. Peers search for objects by sending query messages to their immediate neighbors. Those queries are evaluated locally in each peer and in case there are matching objects, results are returned to the searching peer. When a query arrives, a peer searches its local synopsis for results and also routes the query to those of its neighbors whose synopses have the closest match. Figure 1 shows an example of our system's operation.

3 Architecture

Our protocol includes the following main steps:

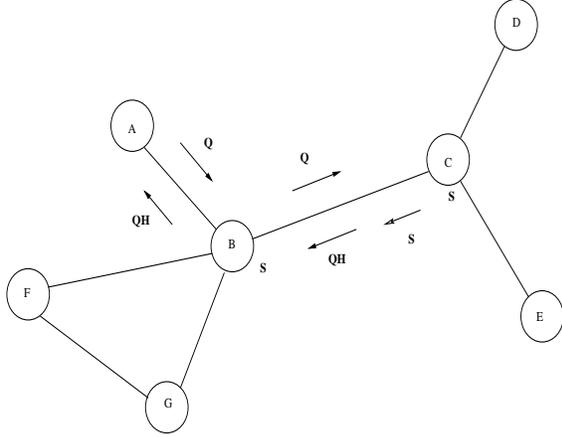


Figure 1: System operation example. According to its criteria, peer C propagated its content synopsis S only to peer B. B based on S was able to route peer A’s query Q only to C, and the result QH is routed back to A.

1. Each peer creates a synopsis of its local content, using a Bloom filter.
2. When a peer is connected to the system, it propagates its local content synopsis to its neighbors.
3. Each peer stores the content synopses of its neighbors.
4. Each peer also maintains statistics regarding its neighbors. Specifically, it keeps track of the number of queries it has received from each of its peers, and of the number of local hits these queries have generated. It also counts the total number of queries it has received.
5. Every time the total number of received queries reaches a certain threshold (currently 100 messages), a peer sends its local content synopsis to peers that are selected adaptively. Currently, the synopsis is sent to the neighbor that has sent the most queries, and to the one whose queries generated the most local hits.
6. When a peer receives a query, apart from searching its local content, it also searches the content

synopses of its neighbors. It forwards the query to all the peers whose synopses state that they contain the requested object. Only if the object is not found in any content synopsis, is the query forwarded to all the neighbors.

The searching in our protocol differs from the Gnutella traditional flooding-based search algorithm, in that the queries are routed –as far as possible– only to peers that seem to have the requested objects. The content synopses propagation in our protocol differs from a traditional Bloom filter propagation, in that the content summaries are not propagated to all the neighbors, but each peer decides which peer to provide with an update of its content synopsis, adapting to the load he receives from the peers. Frequent updates are especially useful for applications where the content of the peers changes rapidly.

4 Simulation Results

We have compared our algorithm for adaptive data propagation (ADP) with an algorithm where each peer simply propagates a Bloom filter of its content to all its neighbors (BF), and with the Gnutella searching algorithm (GNT).

In order to be able to evaluate systems of thousands of peers, we have implemented the algorithms on top of the Gnutella [4] unstructured peer-to-peer network using the NeuroGrid simulator [5]. Neurogrid is scalable, since it simulates the protocols at message- and not at packet-level. The implementation was done in approximately 2500 lines of Java. The main classes include the following:

- ADPGnutellaNetwork
- ADPGnutellaNode
- ADPGnutellaMessageHandler
- ADPGnutellaNetworkStatistics
- ContentSynopsis
- ContentSynopsisMessage
- BloomFilter

We used counting Bloom filters, so that hashes landing in the same position are counted. We used a 4-bit counter, filters of 10bits length, and 4 hash functions. For the hash functions we used SHA-1, a cryptographic message digest algorithm that hashes arbitrary length strings to 160bits. The hash functions are built by dividing the SHA-1 output into groups. We used the following simulation parameters:

- 300 possible Documents
- 400 possible Keywords
- 30 Documents per Node
- 1 Keyword per Document
- 50 Maximum Connections per Node
- TTL 7

Our metrics try to measure both the utilization of the resources, and the efficiency of the system, and therefore include the traffic overhead (number of messages), and the query success rate. Our results are averages of 5 measurements.

Figures 2 3 show that by propagating the content synopses adaptively we can route queries more efficiently, maximizing the hits. Furthermore, as figure 4 shows, the amount of queries that are propagated by error, due to false positives in the Bloom filters, is becoming smaller, since the content synopses are more accurate.

Figure 5 shows that the traditional flooding maximizes the number of objects found. However this comes at a high cost in message transfers, as figure 7 shows. By comparing the adaptive Content Synopses Propagation to the propagation of Bloom filters to all the neighbors, we conclude that the number of messages sent is lowered when propagating adaptively (figure 8), whereas the amount of objects not found is not that smaller (figure 6).

Examining the number of nodes reached during a search lets us conclude again that GNT reaches a lot of nodes unnecessarily (figure 9), whereas ADP reaches even less nodes than BF (figure 10). Moreover, using the Bloom filters (ADP, GNT), the first

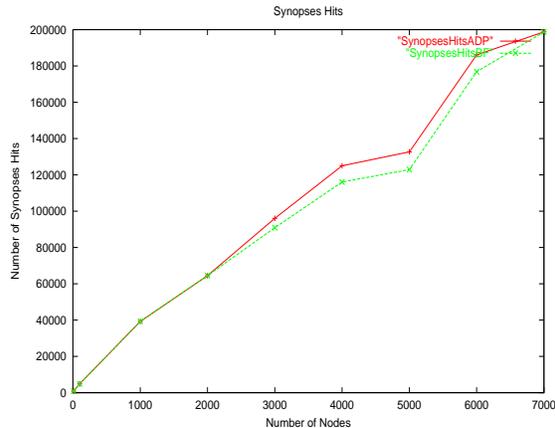


Figure 2: Queries found in neighbors' content synopses.

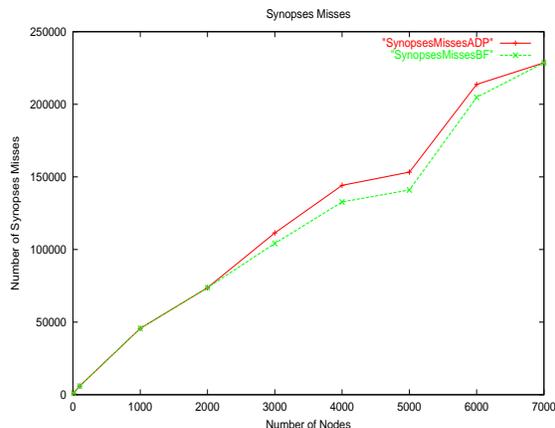


Figure 3: Queries not found in neighbors' content synopses.

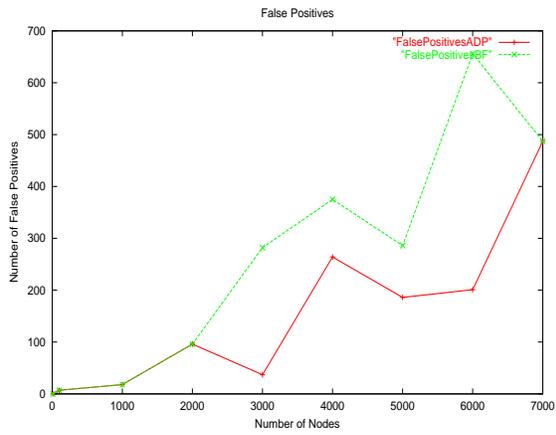


Figure 4: Queries falsely propagated.

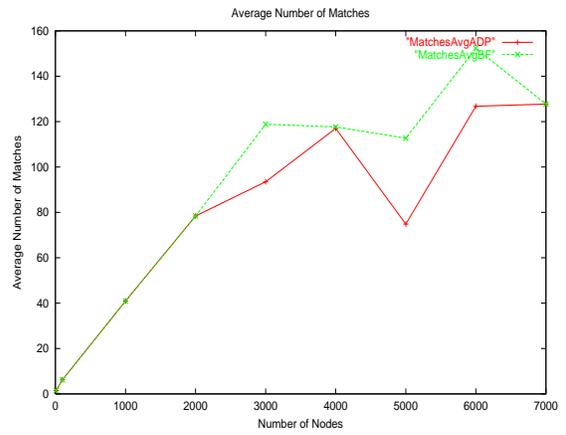


Figure 6: Number of matching objects found for a search.

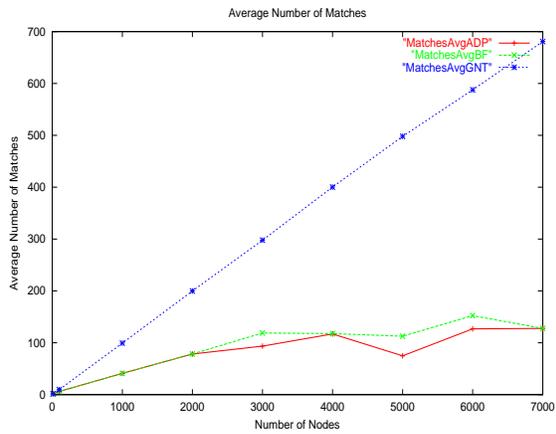


Figure 5: Number of matching objects found for a search.

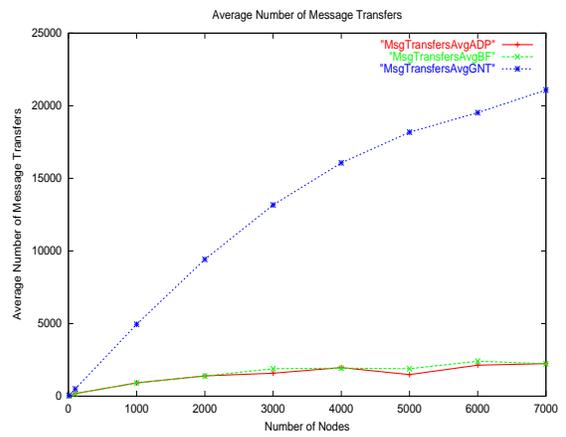


Figure 7: Number of messages sent during a search.

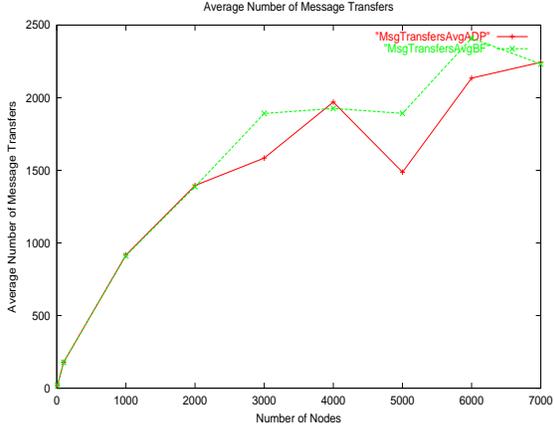


Figure 8: Number of messages sent during a search.

hit to a query is achieved a little bit earlier (figures 11 12).

Figures 13 14 show that –as expected– flooding (GNT) discovers a higher proportion of all possible matches than ADP and BF. However, as figures 15 16 show, this comes at the high cost of message transfers, which becomes enormous as the number of nodes increases.

5 Related Work

Several mechanisms have been proposed to improve searching and update propagation in peer-to-peer networks.

In Planet-P [2] two data structures are replicated globally: A membership directory and a compact content (term-to-peer) index. Members gossip about changes to keep these data structures updated and loosely consistent. Gossiping is done by pushing rumors to random peers and by pulling (anti-entropy) information from random peers. A content ranking algorithm based on the vector space ranking model is also used, to find only highly relevant documents to a query. The set of terms in each peer’s local index is summarized using a Bloom filter. The global index is used to find peers that have a term, and then the local index is used to return the specific documents. Gossiping is done when a Bloom filter changes, when

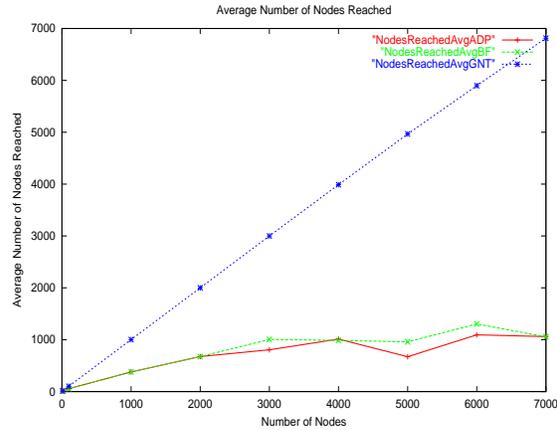


Figure 9: Number of nodes reached during a search.

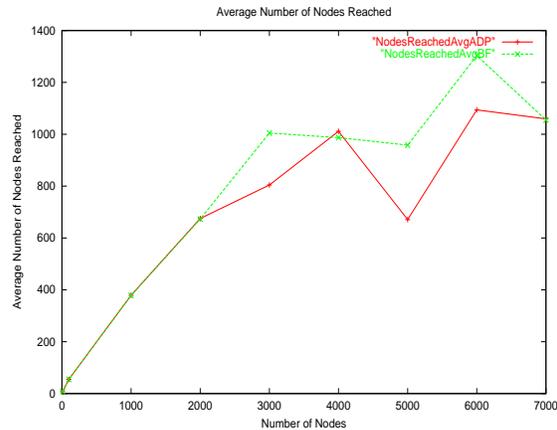


Figure 10: Number of nodes reached during a search.

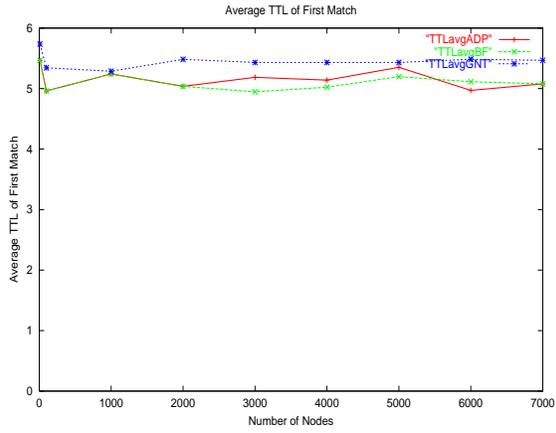


Figure 11: The TTL of the first message that found the first match (i.e. how many hops before the first hit).

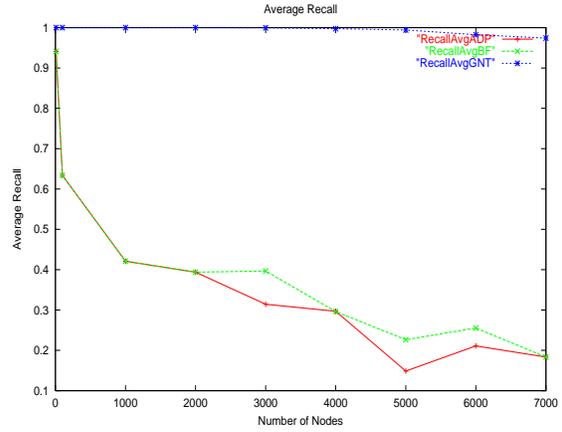


Figure 13: Proportion of all possible matches that was actually discovered.

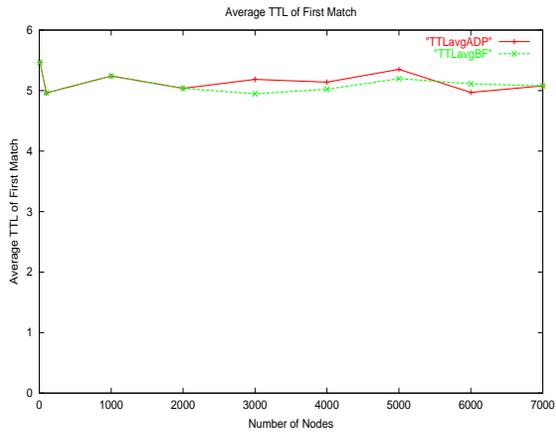


Figure 12: The TTL of the first message that found the first match (i.e. how many hops before the first hit).

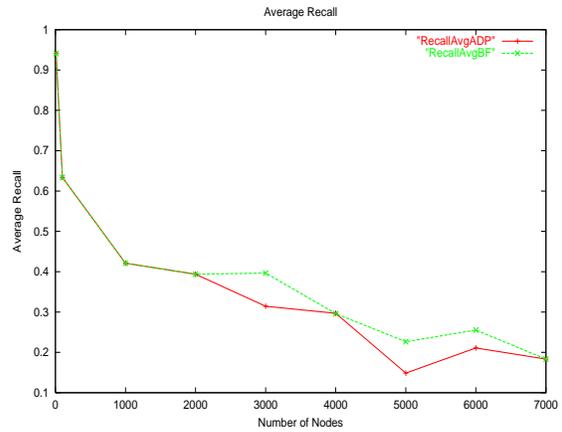


Figure 14: Proportion of all possible matches that was actually discovered.

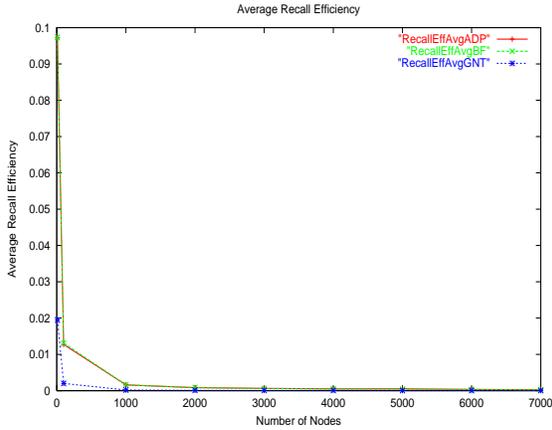


Figure 15: Average Recall divided by the number of messages transferred.

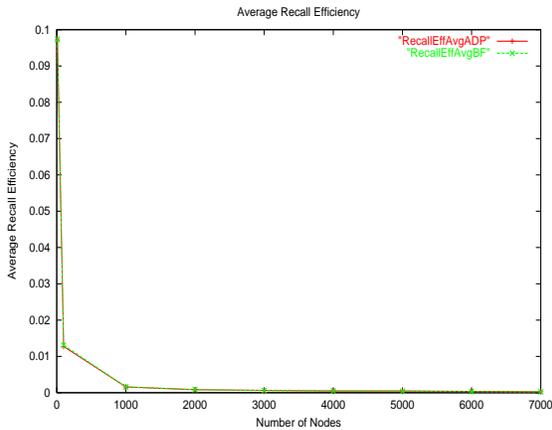


Figure 16: Average Recall divided by the number of messages transferred.

a new member joins, when a previously off-line member rejoins, but not when a member leaves, temporarily or permanently. The cost of storing and maintaining the global data structures makes the system unsuitable for users with modem-speed connections, low storage capabilities, or for networks of more than some thousand peers.

P-Grid [3] uses a hybrid push/pull rumor spreading algorithm, that offers probabilistic guarantees, instead of ensuring strict consistency. A new update is pushed by the initiator to a subset of peers that are affected by it, because they have the original version of the data item, and is further propagated by them. Peers that have been disconnected, that have not received updates for a long time, or that have received a pull request but are not sure if they have the latest update, pull updates from one or more other peers. Two parameters, the probability of forwarding an update, and the fraction of the total replicas to which peers initially decide to forward an update, are important for the spreading of the rumors.

CUP (Controlled Update Propagation) [10] is used for maintaining caches of metadata for locating content. A node receives and propagates updates only when it has personal economic incentive to do so, in other words only when the benefit from the propagation outweighs the cost of it. The investment return is secured when a node can answer queries using the stored metadata, instead of having to further forward them. Each node decides whether to register for receiving and propagating updates for an item according to popularity (based on the number of queries received for that item)-based incentives, either probabilistic, or log-based, also taking into account its workload and/or network connectivity. Query coalescing is utilized, enabling nodes to push only one query when receiving multiple queries for a stale item. Updates are either deletes, refreshes, or appends. Each node keeps for each key a flag indicating whether the node is waiting for a response to a query for that key, as well as an interest bit vector, that indicates which neighbors are interested in receiving updates for that key.

Breadth and Depth Bloom filters have been used to make searching of hierarchical data more efficient. In [6] each node maintains a local filter, summarizing

the documents stored locally, and one or more merged filters, summarizing the documents of its neighboring nodes. When a query reaches a node, the node first checks its local filter and uses the merged filters to direct the query only to those nodes whose filters match it. To summarize hierarchical data, such as XML documents, Breadth and Depth multi-level Bloom filters are introduced. Following a hierarchical organization, specifically one that is based on the similarity on content (on filters) and is controlled by a similarity threshold, enables efficient query routing. Effort is made to propagate only the minimum update information (just the differences in the merged filters).

In QRP (Query Routing Protocol) of RFC-Gnutella 0.6 [4] ultrapeers are responsible for filtering queries and only forwarding those to the leaf nodes that are most likely to have a match. This filtering is done by looking the query words through a hash table that is sent by the leaf node to its ultrapeer.

There has also been work on keyword searching in DHTs. In [9] each node in the DHT stores a list of nodes that contain a specific keyword. Inverted indices are used, that map each word found in any document to a list of the documents that contain that word. The partitioning is vertical, meaning that each node maintains pointers to all the documents that contain a specific word. Bloom filters are used to reduce the bandwidth required to answer “AND” queries (which need the cooperation of more peers’ incremental results to be answered), and to cache document lists. According to the virtual hosts approach that is used, a node participates in the peer-to-peer system as several logical hosts, proportional to its request processing capacity.

Also relevant is work done on filtering and disseminating streaming data [11]. In order to provide updates of highly dynamic, streaming, and aperiodic data, an organization of data repositories is proposed. The repositories are organized hierarchically, with those that have the highest coherency requirements placed closer to the data source. Data updates are pushed down that hierarchy, only to the repositories that require them (according to their coherency requirements). Repositories are placed in a way that their coherency requirements are *just* met,

so that repositories with more stringent coherencies end up serving repositories with more loose coherencies. Back-up parents are used to handle repository or communication link failures. Active back-up parents deliver data with less stringent coherency, reducing the overhead of providing resiliency and enabling the detection of the failure.

There has also been work in caching the results of queries, while partitioning a network in layers. In [12] in addition to a local index, that keeps indices of local files, each peer maintains a response index, which caches the query results that flow through the peer. That type of uniform index caching saves network bandwidth, but does not avoid a large amount of duplicated caching, due to the cache hit overlap between the neighboring peers. Instead of caching query responses in all peers along the returning path, distributed caching attempts to cache the responses only in selected peers. Specifically, peers are separated in groups (layers), and a response is cached in a peer, only if the hash value of the query is the same as the peer’s group ID. Accordingly, a query is forwarded only to neighbors with a group ID that matches the hash value of the query. Following this approach, there exists the possibility of missing matched peers or objects. To avoid these misses, pushing the indices of objects whose hash values do not match a peer’s group ID to matching neighboring peers is also introduced. By using distributed caching and adaptive searching and by dividing the network in layers, search traffic is reduced significantly, while the query success rate is not degraded. However the partitioning in layers is done arbitrarily.

Our work can be seen as an extension to [8] and [7], in that the content summaries that have also been presented there are propagated to peers that are selected adaptively.

6 Conclusions

By propagating content synopses to peers that are selected adaptively, we manage to make the search and retrieval of data in unstructured peer-to-peer networks faster, and more efficient in terms of bandwidth and processing power usage. However, –as expected–

the recall of the flooding-based search is not reached. The main advantage of our protocol, namely the reduction of search messages, is particularly evident in large-scale networks. Moreover, by frequently propagating content updates to the peers that need them the most (which are selected adaptively), instead to all the neighbors, we can keep the number of messages low, and yet route queries accurately. The advantages of our adaptive approach are even more evident in applications where the content of the peers changes rapidly.

Our future work includes the comparison of our push-based protocol to an analogous pull-based, as well as investigating the effect of propagation of content synopses to peers further than the immediate neighbors. Moreover, taking into account more parameters when deciding where to propagate the content synopses might also be interesting.

References

- [1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.
- [2] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities. In *Proceedings of the International Symposium on High Performance Distributed Computing, HPDC*, June 2003.
- [3] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Proceedings of the International Conference on Distributed Computing Systems, ICDCS*, May 2003.
- [4] Gnutella Protocol Development. <http://rfc-gnutella.sourceforge.net/>, 2003.
- [5] S. Joseph. An extendible open source P2P simulator. *P2P Journal*, pages 1–15, November 2003.
- [6] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *Proceedings of the International Conference on Extending DataBase Technology, EDBT*, March 2004.
- [7] A. Mohan and V. Kalogeraki. Speculative routing and update propagation: A kundali centric approach. In *Proceedings of the International Conference on Communications, ICC*, May 2003.
- [8] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding good peers in peer-to-peer networks. In *Proceedings of the International Parallel and Distributed Computing Symposium, IPDPS*, April 2002.
- [9] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the International Middleware Conference*, June 2003.
- [10] M. Roussopoulos and M. Baker. CUP: Controlled update propagation in peer-to-peer networks. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.
- [11] S. Shah, S. Dharmarajan, and K. Ramaritham. An efficient and resilient approach to filtering and disseminating streaming data. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, September 2003.
- [12] C. Wang, Y. Liu, and P. Zheng. Distributed caching and adaptive search in multilayer p2p networks. In *Proceedings of the International Conference on Distributed Computing Systems, ICDCS*, March 2004.