# Efficient Data Dissemination in Overlays [*]

Dung Vu, Thomas Repantis and Vana Kalogeraki
Department of Computer Science and Engineering
University of California, Riverside, Riverside, CA 92521
Email: {dungv, trep, vana}@cs.ucr.edu

## Abstract

*In this paper we propose adaptive data dissemination algorithms for intelligently routing search queries in a peer-to-peer network. In our mechanism nodes build content synopses of their data and adaptively disseminate them to the most appropriate nodes. Based on the content synopses, a routing mechanism is being built to forward the queries to those nodes that have a high probability of providing the desired results. Our simulation results show that our approach is highly scalable and significantly improves resources usage by saving both bandwidth and processing power.*

## 1. Introduction

The explosive growth of rich-media online content, such as audio, video, news articles, images, and documents has created new challenges for real-time collaboration among multiple users in large-scale distributed environments. This, coupled with advances in the networking, processing and storage capabilities of personal computers has signaled the emergence of peer-to-peer (P2P) systems as a platform for providing and receiving data and services. In a peer-to-peer system, peers form an overlay over the physical network and employ their own location and routing mechanisms and maintain soft state information about other peers. The peers can be geographically distributed, heterogeneous in their resource capabilities, and dynamic in their participation in the system. Peer-to-peer systems have been used with great success for storing and sharing data [8] as well as for performing distributed computations. Some of their attractive features include cost effectiveness (by aggregating existing resources), increased autonomy (by self-organizing), improved scalability (due to the absence of a central coordinator's bottleneck), and reliability (due to lack of a single point of failure).

Several research efforts on Distributed Hash Tables [19, 21] have focused on imposing a structure in the peer-to-peer overlay by employing different algorithms to assign object keys to nodes to guarantee key retrieval in logarithmic time. Even though structured overlays achieve object retrieval in bounded time, they have been inherently limited in other ways [3]: They do not support complex keyword-based queries without constraints on data placement, they do not take peer heterogeneity into account, and do not handle robustly network dynamics, like massive peer arrivals, departures, or failures. Several efforts have been made to address all of the above issues [3].

In this work, we focus on unstructured overlay networks, in which objects can be located at random nodes, and nodes are able to join the system at random times and leave it without a priori notification. Our motivation stems from the fact that unstructured overlays have been deployed and are being used by millions of Internet users. However, in an unstructured topology several design issues arise, one of the most challenging ones being the efficient search and retrieval of data or services [1]. The major issue is that no central manager can have an accurate global view of the system's contents. The problem is complicated further by the fact that the environment is dynamic and heterogeneous. Peers join, leave, and fail without a priori notification and have very different and restricted processor, storage and communication capabilities. Finally, in a large-scale peer-to-peer network, the amount of traffic generated by queries can be overwhelming.

Traditionally, search in unstructured peer-to-peer networks has been performed based on keyword queries, by flooding the network with messages and propagating the search query hop-by-hop until the desired answer is found. The problem with this approach is that it fails to take into account the probability of a node to be able to provide the requested 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.

Building upon this breadth-first search protocol, sev-

[*]This research was supported by NSF Award 0627191.

[1]We will be using the term "object" to refer to both data and services.

eral efforts have focused on improving the efficiency and scalability of searching in unstructured overlays, including random walks [13], routing indices [5] and query caching [24, 25]. Using random walks, a peer randomly forwards its query to a subset of its neighbors, which repeat the same process. In [4] biased random walks are combined with flow control and topology adaptation to take into account the heterogeneity of the peers. Using routing indices, queries are guided towards the direction of the requested object, as each peer maintains statistics which indicate how many objects are reachable through each neighbor. Efforts on query caching use the keywords of the queries to compute the similarity of the query message to previously seen queries, to probabilistically forward the query to only a subset of the nodes. These rely on knowledge collected locally at the peer by monitoring the messages propagated in the network. Caching the results of queries, while arbitrarily partitioning a network in layers has also been proposed [24].

Content summarization has been proposed as a technique for reducing traffic and alleviating hot-spots in unstructured peer-to-peer systems. Allowing nodes to maintain summaries of remote peers' content enables them to guide queries to peers more likely to provide relevant answers. Bloom filters [2] are a probabilistic data structure for content summarization that has been used in peer-to-peer environments [6, 11, 14, 19]. In Planet-P [6] gossiping is used for propagating content indices across peers. In OceanStore [19] attenuated Bloom filters are propagated to improve searching in a structured peer-to-peer network. Breadth and depth Bloom filters have also been used for summarizing hierarchical data structures such as XML documents in a hierarchically organized network [11]. In our previous work [18] we have explored strategies for disseminating Bloom filter content summaries in unstructured peer-to-peer networks.

In this work we target the problem of data dissemination in unstructured, decentralized peer-to-peer networks. We combine a backbone-based topology with content summarization to enable peers to locate relevant content faster and with smaller message overhead. Nodes construct Bloom filter summaries of the objects in their local stores and propagate them to highly available and powerful peers. By having access to these summaries, these backbone peers can efficiently guide queries to peers with relevant content, to maximize the probability for a fast reply. We present an experimental study of large-scale networks. Our results illustrate that in comparison to other popular searching techniques, our approach reduces the number of messages sent, as well as the number of peers contacted, and achieves this with small overhead for building the backbone overlay.

The rest of this paper is organized as follows: In section 2 we present our architecture for efficient peer-to-peer data dissemination. In section 3 we present our backbone construction and search algorithms. In section 4 we describe the experimental evaluation of our approach and discuss our results. We review related work in query routing and data dissemination in overlay networks in section 5. Finally, we draw conclusions in section 6.

## 2. System Model

We consider a network of $\underline{N}$ nodes (peers) that store objects. Each peer has a globally unique identifier (e.g. IP:port, or a randomly generated number) and maintains connections with other peers. The network is unstructured, decentralized and self-organizing, meaning that peers make their own decisions regarding to which peers to connect to or to query for objects. The number of connections of a peer can vary and is typically restricted by its resource capabilities. Each object is uniquely identified by the means of intrinsic references [7] which are generated when the object is first inserted in the system. Intrinsic references are based on the hash digest of the object's actual contents rather than its name or location and therefore allow us to create persistent, state-independent, and immutable storage. The mechanisms presented in this paper are orthogonal to the type of search and therefore we just focus on searching by an object's intrinsic reference.

### 2.1. Content Synopses

Each peer uses the *Bloom filter* data structure [2] to build a synopsis of the content in its local store. Assume that peer $p$ has a group of $n$ objects given by the set $S_p = a_1, a_2, ..., a_n$. The Bloom filter that represents the set $S_p$ is described by a bit array $BF_p$ of length $m$, with all bits initially set to 0. We assume $k$ hash functions, $h_1, h_2, ..., h_k$ with $h_i : X \rightarrow 1...m$. Each hash function maps each element of the set $S$ to a value between $1...m$ in a totally random fashion. For each element $s \in S$, the bits at position $h_1(s), h_2(s), ..., h_k(s)$ are set to 1. To determine whether a certain element $x$ is in $S$, we check whether all the bits given by $h_1(x), h_2(x), ..., h_k(x)$ are set to 1. If any of them is 0, then we are certain that the data item $x$ is not in the set $S$. If all $h_1(x), h_2(x), ..., h_k(x)$ are set to 1, we conclude that $x$ is in $S$, although there is a certain probability that we are wrong. This is the case that a Bloom filter may yield a false positive. Our system exploits the probability that a small number of false positives does not greatly affect the performance of our searching mechanism. This fact makes the Bloom filter approach highly suitable for locating objects accurately and fast.

To support the removal of members from the sets represented by the Bloom filters, we use counting Bloom filters. In this approach, a counter is added to each bit in the filter, so that the number of objects that are hashed in the same position is counted.

Each backbone peer stores filters for objects stored in peers in its region, indexed by the peers' identifiers. Hence, to store multiple content synopses, we use multi-level Bloom filters. Notice that the Bloom filter of each level is not merged but appended to that of the previous level. That approach consumes more memory space to store the Bloom filters, but allows us to estimate the location of a larger number of objects more accurately.

## 3. Backbone

We propose a novel two-level search approach, which is a combination of a broadcast-based search in the local region and a network-wide search with the support of a network backbone. The backbone is a connected overlay network of peers identified through a self-election process. The backbone peers are elected based on their capabilities for handling queries, which include available bandwidth and processing power, as well as uptime. Using Bloom filter data summaries, the backbone stores content synopses of the data in the entire network. Our two-level search mechanism takes advantages of a locally confined broadcast search to quickly find popular objects available in a peer's vicinity, and of a search on the backbone overlay to find distant rare objects. The backbone search takes advantage of the content synopses to avoid a broadcast storm in the entire network.

**Construction of Backbone.** A peer must demonstrate that it has sufficient resources and stability before it can qualify for becoming a backbone node for a region. The requirements for becoming a backbone node are: i) sufficient bandwidth, ii) sufficient processing power, iii) sufficient uptime.

Backbone nodes are elected among neighbor peers in their vicinity using the following distributed protocol. In the beginning, each peer $p$ assumes it is the best candidate to be elected as a backbone node. Peer $p$ broadcasts a message advertising its capacity (bandwidth, processing power, uptime) to challenge its neighbor peers $q$. At the same time, peer $p$ also compares all incoming advertised capacities from neighbor peers $q$ with the best capacity $p$ has seen so far. If the advertised capacity of an incoming message $w$ is better than the one peer $p$ knows, $p$ elects the peer $e$ that generated the winning message $w$ as its elected backbone node and propagates the winning message to all of its neighbor peers $q$. Otherwise, peer $p$ drops the message. If the same message arrives later again from other paths, peer $p$ does not propagate it again. Passing through peers, the winning messages accumulate the connection degrees of each peer. Using this accumulative connection degree (ACD) value, receiving peers decide to stop or continue

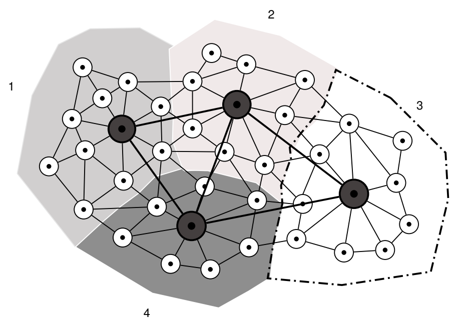


**Figure 1. Backbone Overlay Network: large black nodes represent backbone nodes, white nodes represent peers. Regions are in different colors.**

propagating the winning messages. To limit the number of peers the winning messages will travel, peers stop propagating the winning messages when the ACD value $a$ the winning messages are carrying exceeds a threshold $th$.

The threshold value $th$ determines the number of client peers having the same elected backbone node. The choice of the threshold value provides a tradeoff. The larger the threshold $th$, the larger the number of client peers with the same backbone node. A large number of peers having the same elected backbone node will impose higher query load to that elected backbone node. Furthermore, in case of backbone failures, a larger number of nodes will be affected by a single backbone node failure. A larger threshold $th$ value will also cause a higher overhead during the reelection process, since the election-related messages travel for a longer time. On the other side, a small number of peers having the same elected backbone node will form a large backbone network. In that case, search on a large backbone will not bring much benefit over the search on the entire network. In the experimental evaluation of section 4 we explore various ACD threshold values $th$, to explore the proportion between backbone nodes and client peers.

**Forming a connected backbone overlay.** Peers check if their immediate peers have the same elected backbone node. If not, these peers are at the border of two neighbor regions. Peers at the border inform their backbone nodes about their neighbor backbone nodes. Backbone nodes of neighbor regions form a connected backbone network. Besides its backbone node, each peer maintains a connection to a neighboring backbone node, used as a backup. The advantage is that, if the primary backbone node fails, the backup node takes over to propagate search queries and replies. Figure 1 shows an example of a peer-to-peer network and its connected backbone overlay network of 4 backbone nodes. The figure shows a network that is divided into 4 virtual regions, and there is one backbone node for each region.

**Content synopses dissemination in backbone.** Since the network is dynamic and self-organizing, nodes may leave or join independently. The system must be able to disseminate content synopses to reflect such changes in the connections. Moreover content synopses must be updated whenever an object is added, deleted, or its contents have changed. In our system, updated content synopses are generated in two cases:

- When a peer detects an update at the local repository (content changes) of objects (new objects are obtained, existing objects are deleted or new versions of existing objects are created).

- When a peer detects an incoming or withdrawn peer connection (connection establishment or drop).

*1) Content changes.* When the content of a node is updated, a new content synopsis is disseminated by the peer. To minimize the traffic in the network our approach (1) does not generate an update unless the contents of the peers have changed and (2) groups individual Bloom filter updates into group updates to propagate them to the peers.
*2) Connection changes.* As nodes in the peer-to-peer network disconnect or reconnect with new peers, the peers in their vicinity change. Therefore connections are dropped and connections with new neighbors are established.

Peers may disconnect from the system either intentionally or due to a failure. When a peer permanently disconnects from the network, neither the content synopses of other peers stored in it, nor its content synopsis stored in other peers will be useful anymore. Its immediate backbone peers will sense the disconnected peer and all the relevant content synopses will be removed after a time threshold $th_{dis}$. In addition, a `DISCONNECTED` message will be sent to the non-immediate peers to remove their corresponding content synopses.

The protocol used for connecting a new peer to the network is analogous to the ultrapeer negotiation utilized in Gnutella 0.6 [8]. When a new peer joins the network, it connects to the backbone node of its geographical domain, or to a random peer who redirects it to the backbone node. If the backbone node has available bandwidth and processing power, it accepts the peer in its region, and adds it to the list of potential backbone nodes, if it qualifies. If the backbone node has reached the maximum number of processors it can support, it accepts the newcomer as a new backbone node if it qualifies, otherwise it redirects it to a backbone node of another region.

When a backbone node disconnects, the backup backbone node takes over to handle queries from the peers. It also activates the election process to re-elect a new backbone node. The re-election process involves only peers in the region; this is locally confined so that it does not cause a significant traffic overhead.

### 3.1. Two-level Search

**Local Search.** In the first level search, a peer broadcasts its query to all peers in its vicinity, within a small number of hops. The number of hops a query will travel is controlled by its Time-To-Live (TTL). If the object is found, the search is complete, and no backbone search is needed. Otherwise, the search proceeds to the second level, which is the search in the backbone.
**Backbone Search.** In the second level search, when a query comes at a backbone node, it is checked against the content synopses stored at this backbone node. If the queried object is not found in the content synopses, the query is propagated to the next backbone node, using a depth-first traversal. This traversal reduces the traffic on the backbone, since it does not use a broadcast mechanism to propagate queries. While depth-first traversal saves the backbone from a message flood, it may increase the latency of the search. However, since the backbone overlay has a much smaller size compared to the entire network and because it consists of peers with high bandwidth capabilities, this disadvantage should not be significant.

When an object is found, it is added to the local content of the peer that originated the query. The peer then updates its content synopsis and sends the updated version to its backbone node. This will increase success probability when the object is queried again. If the query reaches all backbone nodes without a positive answer, the requested object is reported as not found. When an object is reported as not found in the content synopses of the entire backbone, the object does not exist in the network. On the other hand, false positive results may occur due to the use of Bloom filters in the content synopses. This means that the search in the content synopses may give a positive result, even though the object does not actually exist in the network.

The more peers are covered by the local search, the less query load is imposed on the backbone. However, the more peers covered by the local search, the more messages are generated due to local broadcasts. To investigate this tradeoff we explore the performance achieved with different TTL values for local search in the experimental evaluation of section 4. TTL controls how many nodes are contacted in the first level search. We identify TTL values that produce good results in our peer-to-peer network.

## 4. Experimental Evaluation

To investigate the performance of our approach we run extensive simulations on the Neurogrid simulator [10] using the Gnutella [8] P2P communication protocol. We use

GT-ITM network generator [23] to generate the peer-to-peer network topology. We have ran extensive experiments using 3 different network sizes, 4 different object densities, 7 local search radius. Each run used 5000 queries. The experiments were run on a Xeon dual processor with 6GB of RAM and 64 bit Linux. Simulation parameters and experimental schemes are presented in table 1 and table 2.

In our implementation we used counting, multi-level Bloom filters. To create the hash functions used in generating the Bloom filters, similarly to [22], we took advantage of a cryptographic message digest algorithm (SHA-1 [15]) and of its property of pseudo randomness. More specifically, we used SHA-1 to hash strings of arbitrary length, representing the peers' content, to 160 bits. We then built the hash functions by dividing the SHA-1 output into smaller sets of bits. In the experiments we used Bloom filters 10 bits long, 4 hash functions, and a total of 2000 unique objects on the nodes. We selected those Bloom filter parameters as optimal, after conducting experiments, which we omit due to lack of space.

| Parameters | Values |
|---|---|
| Network sizes (nodes) | 2,000; 5,000; 10,000 |
| Maximum connection degree | 20 |
| Network topology | Random |
| Accumulative Connection Degrees Thresholds | 20, 30, 40, 50, 60 |
| Maximum Number of initial unique objects | 500,000 |
| Number of object replicas per 1,000 nodes | 1, 5, 10, 50 |
| Number of object stored in a peer | 500 |
| Number of queries | 5,000 |
| Phase 1: BroadCast-based Search | Gnutella |
| Phase 2: Backbone Search | Depth-first Traversal |
| Object and Query Distribution Scenarios | Uniform |
| Local search radius (hops) | 0 - 7 |
| Percentage of backbone node losses | 0%, 20%, 40% |

**Table 1. Simulation Parameters**

| Schemes | Nodes | Docs per nodes | total replicas | Unique docs |
|---|---|---|---|---|
| 2K.1R | 2,000 | 500 | 2 | 500,000 |
| 2K.5R | 2,000 | 500 | 10 | 100,000 |
| 2K.10R | 2,000 | 500 | 20 | 50,000 |
| 2K.50R | 2,000 | 500 | 100 | 10,000 |
| 5K.1R | 5,000 | 500 | 5 | 500,000 |
| 5K.5R | 5,000 | 500 | 25 | 100,000 |
| 5K.10R | 5,000 | 500 | 50 | 50,000 |
| 5K.50R | 5,000 | 500 | 250 | 10,000 |
| 10K.1R | 10,000 | 300 | 10 | 300,000 [2] |
| 10K.5R | 10,000 | 500 | 50 | 100,000 |
| 10K.10R | 10,000 | 500 | 100 | 50,000 |
| 10K.50R | 10,000 | 500 | 500 | 10,000 |

**Table 2. Experimental Schemes.**

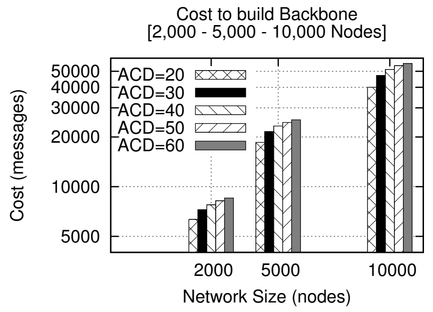
An example of scheme naming convention in table 1, 5K.1R means a P2P network of 5,000 nodes, object density is 1 replica per document for each 1,000 nodes. Since the network size is 5,000, the total number of replicas is 5. 2K.1R - 5K.1R and 10K.1R schemes will have the same object density (1 replica per object for each 1,000 nodes) although the total replicas corresponding to their network sizes will be different as shown in table table 2. For experiments with Uniform distribution, the above schemes are added with a suffix of Unif., such as 5K.1R.Unif. For experiments with local search and backbone search , an additional LS and BS suffice is added, respectively, for example, 2K.1R.Unif.BS.

**Creating the Synopsis backbone.** Cost to create the backbone is measured as number of broadcast messages needed to elect backbone nodes for entire P2P network. To observe how accumulative connection degree (ACD) parameter would efficiently build balanced backbone overlay network we run experiments with different ACD threshold values on three different network sizes: 2,000 - 5,000 and 10,000 nodes. Figure 2 shows the cost of building the backbone overlay and average region sizes with different ACD thresholds. We observe the followings:

- The larger the ACD threshold values are, the further the election messages will travel, and the larger region sizes that the elect backbone nodes will be responsible.
- Region sizes are proportional to ACD threshold values and independent with networks sizes. Thus, we can use ACD threshold values to establish backbone network with designated average region size. With ACD values of 50 to 60, the average region size is 9 to 10. Since the network is randomly created where peers have arbitrary connection degrees, there is a significant number of regions that have 2 peers. We use ACD threshold value of 50 to build backbone for the entire experiments. Each region has an average size of 10 peers.

**Cost of The Two-Level Search.** The cost of search is measured by a number of query messages traveling between peers to search for requested object. If an object is found locally with a local broadcast-based search, the cost only consists of the cost of the local search. If a query fails to find an object using local search (within a search radius) and the query is forwarded to the backbone search, the total cost will consist of the cost of the local search and the cost of the backbone search. The more hops or the larger local search radius, on which the local search is conducted, the more expensive the local search cost and the overall cost is. To observe the interaction between local search and backbone search, and find an efficient combination we run experiments of local search with different local search radius from 0 to 7 hops, corresponding to equivalent TTL values. Figure 3 shows the results of the experiments with Uniform distribution. In figure 9, we only plot result of 2K.1R.Unif.LS

[2]Due to RAM constraint, in 10K.1R scheme number of documents per node and total number of unique documents are adjusted.



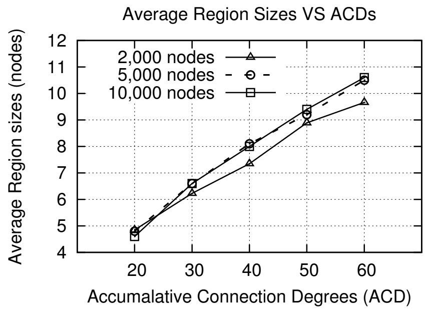


**Figure 2. Cost to build backbone. Average regional sizes are proportional to values of ACD and independent of network sizes.**

(2,000 node network, 1 replica, uniform distribution, local search). Local search costs of different densities are different, however, they are indistinguishable in logarithmic scale. Therefore, we just plot for one scheme to represent the rest. We observe the followings:

- When the number of local search radius is 0 hop, there is no local search but pure backbone search. In this case, the total cost only consists of the backbone search. When the number of local search hops is from 1 hop and above, the overall cost is a total cost of local search and backbone search. When the number of search hops is 7 or above, no backbone search is performed, but only pure broadcast-based search when all queries are solved using local search. In this case, the cost of local search reaches maximum while the cost of backbone is reduced to zero.
- The higher the object density is, the less backbone search cost is required. The scenarios of 50 replica (50R) always have the least expensive costs.
- For low object densities (1 or 5 replicas per object each 1,000 nodes), a small local search radius does not help to reduce the cost on backbone. It is explained that with low object density, objects are thinly spread all over the network and so that fewer objects are available in close vicinity.

Figure 4 shows cost total cost of two-level search including local broadcast-based search and backbone search for Uniform distribution. We select a typical network size of 5,000 nodes to show experiment results. The other network sizes have similar results. We observe the following:

- Total cost of the two-level search is reduced to minimum when there is only backbone search. This shows advantages of backbone search when queries are propagated along the backbone only, while broadcasting of query messages are avoided. However, this ideal search might be not realistic when the backbone is overloaded by queries.
- When the local search radius is about 2 hops, the total cost is not as minimum as the cost with backbone search only, but the cost is acceptable (with this scenarios, less than 100 messages per query). However, this minor trade-off returns a great benefit when query load on backbone is greatly reduced. With 10 replicas density, query load is reduced up to 40% with Uniform distribution.

## 5. Related Work

The idea of using backbone networks is not new. It has been used in the context of multi-cast or wireless network environments. However, our use of backbone is to improve data dissemination in peer-to-peer overlays.

The Query Routing Protocol (QRP) of RFC-Gnutella 0.6 [8] employs ultra-peers to filter queries and only forward the queries to the leaf nodes that are most likely to have a match. This filtering is done by looking at the query words through a hash table that is sent by the leaf node to its ultrapeer. This scheme differs from our approach in the way backbone nodes are elected. Similar to our scheme is the approach followed by FastTrack, a proprietary protocol used by KaZaA [12] and other file sharing applications: Superpeers with higher networking, storage, and processing capabilities volunteer to maintain meta-data for files located in regular peers. This way queries have to travel only through a network backbone before they reach nodes that can offer results. [26] proposes a design of super-peer network that operates exactly like a pure P2P network. Each node in the superpeer network is a superpeer and is connected to a set of clients. Clients are connected to a super-peer only. In the design $k - redundancy$ is proposed to provide reliability and reduce load on superpeer.

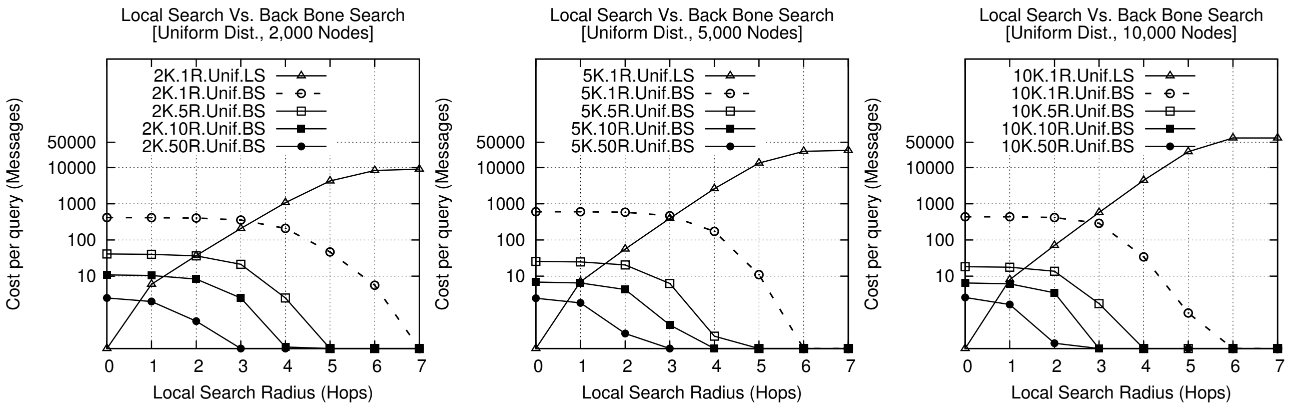


**Figure 3. Backbone Search Cost: Backbone search cost reduces to zero as local search radius increases.**

Similar to our approach, a scheme to minimize redundant messages through the construction of FloodNet, a tree-like sub-overlay over the existing P2P system is proposed in [20]. Searching is divided into two stages. In the first stage, a message is propagated by using the standard flooding scheme with three or four TTL hops. In the second stage, the message propagating is only conducted across the FloodNet. In [26] a workload model is proposed by analyzing workload on super-peers and entire network to obtain an optimal layer size ratio between superlayer and leaflayer. It also proposes a distributed Dynamic Layer Management algorithm to maintain the optimal layer size ratio and adaptively elect and adjust peers between superlayer and leaflayer. [16] proposes a generic profile-driven distributed data dissemination system for dissemination-based application services such as RSS, multicast-based content distribution. It also presents a tree-oriented optimization framework, which can be customized per application for application-specific data filtering, profile aggregation, and optimization logic. [9] proposes content-abundant cluster-selectively prefetching indices from responding peers (CAC-SPIRP) consisting of two components: (1) CAC, content-abundant inter-connected cluster where peers self-identify, and self-organize themselves providing a pool of popular objects to be frequently accessed by the peer community. (2) SPIRP aims to improve the quality of P2P search. Peers prefetches entire file indices of the related interests of peers who have the same interests.

Similar to our approach, Rhea and Kubiatowicz propose a probabilistic location protocol based on attenuated Bloom filters [19] which improves the latency of locating files. Again, the difference from our mechanism is that they place the objects to specific nodes based on some keys and use these keys to route the requests to the nodes. Furthermore, we investigate different algorithms for disseminating the content synopses. Aspnes et al [1] have shown that there is no need for such global coordination in the network. Our approach has the advantage that it does not impose any structure; we assume that the system is self-organizing, driven only by decisions made locally at the peers.

Our work builds upon [14] and [17]. In [14] the concept of content summarization was introduced, while in the current work we focus on mechanisms for the dissemination of the content synopses, we present more elaborate summarization techniques and discuss performance in highly dynamic environments. One of our criteria for the adaptive selection of the content synopses recipients is the notion of interests, explained in [17].

## 6. Conclusions

In this paper we have investigated adaptive data dissemination mechanisms for peer-to-peer networks. We build a backbone overlay with content summarization that enables nodes to forward queries intelligently only to their peers that are able to provide replies with a high probability. We have simulated peer-to-peer networks of thousands of peers and also verified the robustness of our mechanism under dynamic peer disconnections. We have compared our content-driven routing mechanism to traditional flooding-based searching to find out great savings in query messages. Thus, our approach is scalable and highly efficient in terms of bandwidth and processing power usage for large-scale peer-to-peer networks.

## References

[1] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant routing in peer-to-peer systems. In Proceedings of the 21st ACM

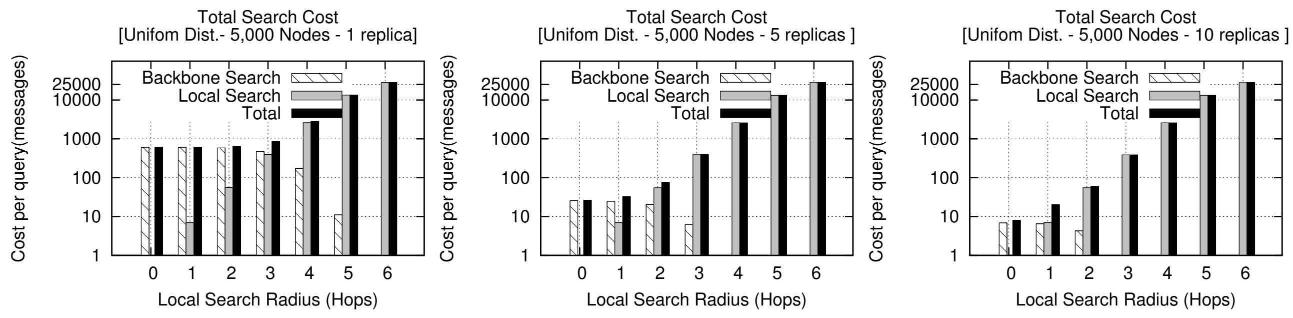


**Figure 4. Total Search Cost**

.

Symposium on Principles of Distributed Computing, PODC, July 2002.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, July 1970.

[3] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM.

[5] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In ICDCS '02, page 23, Washington, DC, USA, 2002. IEEE Computer Society.

[6] 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 12th IEEE International Symposium on High Performance Distributed Computing, HPDC, June 2003.

[7] K. Eshghi. Intrinsic references in distributed systems. Technical Report HPL-2002-32, HP Labs, 2002.

[8] Gnutella Protocol Development. http://rfc-gnutella.sourceforge.net/, 2003.

[9] L. Guo, S. Jiang, L. Xiao, and X. Zhang. Fast and low-cost search schemes by exploiting localities in p2p networks. J. Parallel Distrib. Comput., 65(6):729–742, 2005.

[10] S. Joseph. An extendible open source P2P simulator. P2P Journal, pages 1–15, November 2003.

[11] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In Proceedings of the 9th International Conference on Extending DataBase Technology, EDBT, March 2004.

[12] J. Liang, R. Kumar, and K. Ross. Understanding kazaa, 2004.

[13] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In ICS '02: Proceedings of the 16th international conference on Supercomputing, pages 84–95, New York, NY, USA, 2002. ACM.

[14] A. Mohan and V. Kalogeraki. Speculative routing and update propagation: A kundali centric approach. In Proceedings of the 2003 IEEE International Conference on Communications, ICC, May 2003.

[15] National Institute of Science and Technology. Secure Hash Standard (SHA1). Federal Information Processing Standard (FIPS) 180-1, April 1995.

[16] O. Papaemmanouil, Y. Ahmad, U. Çetintemel, J. Jannotti, and Y. Yildirim. Extensible optimization in overlay dissemination trees. In SIGMOD '06, pages 611–622, New York, NY, USA, 2006. ACM.

[17] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding good peers in peer-to-peer networks. In Proceedings of the 2002 International Parallel and Distributed Computing Symposium, IPDPS, April 2002.

[18] T. Repantis and V. Kalogeraki. Data dissemination in mobile peer-to-peer networks. In MDM '05: Proceedings of the 6th international conference on Mobile data management, pages 211–219, New York, NY, USA, 2005. ACM.

[19] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In Proceedings of IEEE INFOCOM 2002, June 2002.

[20] L. G. Song Jiang and X. Zhang. Lightflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In ICPP'03: Proceedings of the 2003 International conference on Parallel Processing, 2003.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[22] The XLattice Project. http://xlattice.sourceforge.net/, 2005.

[23] T.Megan and Z.Ellen. Generation and analysis of random graphs to model internetworks. Technical Report GIT-CC-94-46, Georgia Institute of Technology, 1994.

[24] C. Wang, L. Xiao, Y. Liu, and P. Zheng. Distributed caching and adaptive search in multilayer p2p networks. In Proceedings of the 24th International Conference on Distributed Computing Systems, ICDCS, March 2004.

[25] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. Information retrieval in peer-to-peer systems. IEEE CiSE Magazine, Special Issue on Web Engineering 2004.

[26] Z. Zhuang. Dynamic layer management in superpeer architectures. IEEE Trans. Parallel Distrib. Syst., 16(11):1078–1091, 2005. Member-Li Xiao and Member-Yunhao Liu.