Abstract: In order for peer-to-peer (P2P) content sharing network to be scalable, it is imperative to efficiently route queries through the network. Semantic based search should, with as little messages (traffic) as possible, return just those relevant documents stored throughout the network thus achieving precision and recall values comparable to those of correspondent centralized system. In this article we propose protocols for self-organizing P2P network that arranges links between peers according to peer's content. In proposed network peers organize themselves into "semantic communities" but without losing links to other semantic communities. Proposed network has no prior knowledge of the semantics of documents that are to be stored in the system.

Key-Words: Peer-to-peer, content-based search, information retrieval, algorithm

1 Introduction
With the advent of Napster we have witnessed extraordinary expansion of interest in peer-to-peer content sharing systems both in general population and in scientific community. With time, better and more efficient protocols (networks) for content sharing have been developed (e.g. Gnutella, eDonkey, BitTorrent). However, widely adopted peer-to-peer protocols for content sharing only allow for metadata searches (file name, size, type, etc.). Peer-to-peer networks that enable content-based searches are still a subject of active research. The ones that have been developed so far are usually divided into structured and unstructured. In unstructured networks peers are unaware of content in neighboring peers which coerces them into less-effective query routing (e.g. Gnutella flooding) resulting in poor network scalability. Structured P2P networks overcome scalability issues but incur complex protocols that are not suitable for highly transient peers typical for P2P systems [10]. Also, structured P2P networks usually have to maintain high-dimensional DHTs (Distributed Hash Tables) that reflect semantic space of documents stored in the system. Such DHTs may be inappropriate for the newly arrived documents whose semantics significantly differ from those of documents that were taken into account during construction of DHT. Along those lines, if we wanted to construct an initially empty network and offer it to the general public to share arbitrary documents we'd have no documents to sample and construct the semantic space.

That was the motivation for construction of P2P network that allows for semantic-based queries without prior knowledge of documents that will be stored throughout the network. Self-organizing P2P network that arranges links between peers according to their content is proposed. In such a network peers organize themselves into "semantic communities". Every peer represents its content with a set of vectors and content likeness is determined as vector likeness. It is assumed that peers (i.e. users) sharing documents of certain topic will most likely search for similar documents (e.g. someone who is sharing papers in the field of computer science is more likely to search for similar papers than e.g. biology papers). Of course, it is entirely possible for user to search for something semantically completely different – therefore it is important not to lose links to other semantic communities.

2 Problem Formulation
Textual documents can be represented and stored as data objects in P2P system. More precisely, a document is represented as an n-dimensional vector, namely Semantic Vector or Feature Vector. Each element in the vector represents the importance of a term in the document. The importance of each term is computed using TF*IDF (term frequency * inverse document frequency) scheme [1]. A term in the document is considered more important if it is used often in that document (TF) and used seldom in other documents in the collection (IDF). During the search process, documents are retrieved according to the similarity of the query vector (which can also be a full-blown document) and document vector. Prevailing measure of similarity is the cosine of the angle between the vectors. If the vectors are normalized, cosine of the
angle can be computed as the inner of product of two vectors:
\[
\cos(Q,D) = \frac{Q \cdot D}{|Q||D|} = \sum_{i=1}^{n} q_i \cdot d_i
\] (1)

This model, in which documents are represented as vectors, is referred to as Vector Space Model (VSM). VSM suffers from synonymy, polysemy and noise in the documents. LSI (Latent Semantic Indexing) [2] technique has been proposed to overcome these issues. LSI uses SVD (singular value decomposition) [1] to transform a high-dimensional VSM vector to a lower-dimensional semantic vector by projecting it into a smaller, semantic, subspace. In summary, both VSM and LSI represent documents as vectors and use cosine of the angle between the vectors to represent their similarity.

Searching for a document in a P2P environment in Gnutella fashion could be done by flooding the neighborhood with query vector but that approach has proven to suffer from scalability issues. Also, since documents are randomly populated (with respect to semantics) it is difficult to achieve good retrieval properties (precision and recall). Although a number of P2P search techniques for unstructured P2P networks (e.g. [11], [12]) have been proposed, they are based on simple keyword matching, search a large number of nodes or don't scale well. Efforts to improve the search efficiency and network scalability have led to constructing structured overlay networks (e.g. CAN [3], CHORD[4]). These systems support hash-table interface of put(key, value) and get(key) and are extremely scalable as they resolve lookups in log(n) routing hops (for a network of n nodes). On the other hand, they support only exact-match queries. Since then, more sophisticated structured systems have been developed (e.g. [5], [6]) that are both scalable and allow for semantic queries. However, being structured, they all have to form a semantic space, probably (not all papers explain it) by sampling documents that are expected to be shared in the P2P network. Although they work well under such conditions, we believe that this presents a problem in the case when there is no prior knowledge of semantics of documents that will be shared throughout the P2P community. To address that problem we propose an unstructured network that clusters the nodes according to their content. Nodes are clustered based on the assumption that peers mostly search for content similar to their own. Proposed network does not require any prior knowledge of content that will be shared.

3 Self-organizing network
Both pure (Fig.1) and hybrid (Fig.5) P2P networks are presented.

3.1 Pure network
Besides sharing content, every node in the pure network routes messages through the overlay and exchanges overlay network maintenance messages (Fig.1). That is, there is no hierarchy of nodes or nodes performing special functions.

Fig.1 Pure network of 10 nodes

Every node represents its content (documents) with semantic vectors (VSM). Vectors can be computed using global statistics that, as demonstrated in [7], doesn't have to be precise. To reduce the number of vectors, similar documents are clustered together and represented with cluster centroid. Number of clusters (or documents per cluster) is arbitrary – it can be tuned over time or even left up to the user to decide (e.g. a user could mark the spot on the dendrogram) although it shouldn't be set too high (this will be a topic for future research). Such set of vectors representing cluster centroids is called node description. In order for a node to join the network it has to connect to existing node(s). Since there is no central authority a node has to find those nodes on its own. In our research new nodes were connected to random existing nodes in the network, and in the real-word applications some already existing techniques (like GWebCaches in Gnutella) could be employed. Each node maintains two sets of links: family links and other links. Family links are used as a connection to other nodes with similar description and other links are used as a connection to communities of dissimilar nodes. Figure 2 shows separately other links, family links and all links for a network of 50 nodes and 3 semantically very distant communities.

Fig.2 other links + family links = all links (50 nodes)

Initially, each node only has other links. Node will start to populate his family links collection when it receives answers to its queries.

3.1.1 Query routing
When a node wants to search for a content in the network it creates a QueryMessage and routes it through the network according to algorithm in Table 1. Two
bloom filters are used to reduce the number of messages that are transmitted through the community of nodes that match the query (based on the similarity to the query vector). Fig.3 shows a small community of nodes that are all interconnected (links are not drawn for clarity). Since every node maintains a list of processed queries, all messages carrying already processed queries are dropped. Dropped messages are drawn with a dotted line. A worst case scenario is shown in which every node can forward maximum two messages and node f never gets the message. To improve message routing two bloom filters [9] are added to the message. Whenever a node forwards the query it embeds into the message node ids (hashes) of all nodes that it will be sending the message to. Accordingly, when a message is forwarded bloom filter is checked to determine whether a node already received the message. Fig. 4 shows a worst case scenario or query routing with the use of bloom filter. Associated table details information about visited nodes that is carried in the correspondent message. When communities with 100-400 nodes, 20 links per node and maximum forward count 10 were flooded it was found that two 100-bit bloom filters have reduced the number of messages by more than 50%.

Fig.3. Routing without bloom filter, MAX_FW_CNT=2

Fig.4. Routing with bloom filter, MAX_FW_CNT=2

Similarity threshold is embedded in the message and, if the query returns too few results, could be adjusted by user (application) to broaden the search. As shown in Table I, a node compares the query vector both to the family links and other links collection. That way, if a query has reached targeted semantic community probably only nodes from the family links collection will be used to forward the query (depending on the threshold and community a query could even be flooded through the community). On the other hand, if the query is somewhere outside the targeted community then probably a most similar node will be found in the other links collection – hopefully that link will lead to the desired community. If none of the known nodes satisfies the threshold requirement then a minimum forwarding rule is activated: query is forwarded to MIN_FW_CNT (e.g. MIN_FW_CNT=1) nodes disregarding the similarity threshold but message's TTL (time to live) attribute is decreased by one. Thus, a message has only TTL hops to reach the targeted community (probably through a series of other links) but once inside the community TTL value doesn't change. On the other hand, if a node computes that more than MAX_FW_CNT links (nodes) meet the threshold requirement then a maximum forwarding rule is activated: query message is forwarded to MAX_FW_CNT nodes from the set of nodes that satisfy the threshold requirement.

<table>
<thead>
<tr>
<th>routeQuery(qMsg)</th>
<th>FwList = getFwList(qMsg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF qMsg.notDoneHopping()</td>
<td>qMsg.setNodesVisited(FwList)</td>
</tr>
<tr>
<td>qMsg.setPreviousNode(this)</td>
<td>forward query to every node in the FwList</td>
</tr>
<tr>
<td>END IF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>getFwList(qMsg)</th>
<th>FW = getRankedNodesDesc(qMsg, qMsg.sim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF FW.size &gt; MAX_FW_CNT</td>
<td>FWList = pick random MAX_FW_CNT peers from FW</td>
</tr>
<tr>
<td>ELSE IF FW.size &lt; MIN_FW_CNT</td>
<td>qMsg.decTTL()</td>
</tr>
<tr>
<td>qMsg.size = getFirst MIN_FW_CNT peers from FW</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td>FWList = getAll peers from FW</td>
</tr>
<tr>
<td>END IF</td>
<td></td>
</tr>
<tr>
<td>RETURN FwList</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>getRankedNodesDesc(qMsg, simTreshold)</th>
<th>FO = Family U Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW = Ø</td>
<td>FOREACH currNode IN FO</td>
</tr>
<tr>
<td>IF currNode ≠ qMsg.src AND currNode ≠ qMsg.prev</td>
<td>AND qMsg.notVisited(currNode)</td>
</tr>
<tr>
<td>IF curr.emptyDesc</td>
<td>FW = FW U (curr, 0)</td>
</tr>
<tr>
<td>ELSE IF qMsg.queryVector.getSimilarity(curr) ≥ simTreshold</td>
<td>ELSE IF qMsg.queryVector.getSimilarity(curr)</td>
</tr>
<tr>
<td>FW = FW U (curr, qMsg.queryVector.getSimilarity(curr))</td>
<td>END IF</td>
</tr>
<tr>
<td>END IF</td>
<td></td>
</tr>
<tr>
<td>END FOREACH</td>
<td></td>
</tr>
<tr>
<td>FW.sortDescending()</td>
<td></td>
</tr>
<tr>
<td>RETURN FW</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Routing algorithm for pure network

Besides routing (forwarding) query message every node evaluates its collection of documents against query vector. If any of the documents meets the threshold the
node sends a \textit{QueryResponse} message to the node that originated the query. Every time a node receives a \textit{QueryResponse} message it updates its \textit{family links} collection with the responder's node description: nodes are sorted descending based on the similarity with its own description. Node maintains its \textit{other links} collection using \textit{MeetTheOthers} message that it emits from time to time. \textit{MeetTheOthers} message is randomly forwarded through the network. When a \textit{MeetTheOthers} message is instantiated, a TTL value is randomly chosen from a predefined interval. Every node that receives \textit{MeetTheOthers} message responds with \textit{NodeDescription} message (carrying only its own description) to source node and then, if TTL is still greater than zero, forwards received message to only one randomly chosen node (that hasn't been visited) from the \textit{other links} collection.

In the event of node failure, nodes simply discard the links with the nodes that don't respond to their messages and other nodes replace the disconnected ones.

3.2 Hybrid network
In order to reduce the traffic (and increase scalability) Gnutella designers have switched from the initial pure (version 0.4) to hybrid (version 0.6) architecture. Accordingly, we've developed hybrid self-organizing network that distinguishes two kinds of nodes: leaf and ultra nodes. The idea is to put more capable (in terms of bandwidth, availability and processing power) nodes in charge of routing the query messages and network maintenance messages. Such nodes are called ultra nodes. Every ultra node maintains connections to a certain number of leaf nodes. Leaf nodes send queries to their ultra node and have no role in query routing process. However in our protocol leaf nodes do communicate with other ultra nodes in attempt to cluster themselves in the semantic communities. Fig. 5 shows a hybrid network with 4 ultra nodes, each attending to 4 leaf nodes.

Fig. 5. Hybrid network with 4 ultra and 16 leaf nodes

3.2.1 Bootstrapping
The network is setup by linking few ultra nodes. Other ultra nodes attempting to join the network have to find an existing ultra node(s) to link with. The process is analogous to the one in the pure network. Leaf nodes attempt to join the network by sending a \textit{JoinRequestMessage} (Fig. 6) carrying node description to an ultra node. If the ultra node already maintains maximum leaf connections it replies with a negative \textit{JoinReply} message carrying a list of alternative ultra nodes (e.g. u2) the leaf can then try to join. Upon receiving a negative reply the leaf node tries to join another ultra node in the list. If the ultra node hasn't reached maximum number of leaf connections it replies with a positive \textit{JoinReply} message and adds the leaf (node description) to its \textit{other links} collection.

3.2.2 Query routing
In hybrid architecture ultra nodes are responsible for query routing thus shielding the leaf nodes. A leaf node creates a query message and sends it to its ultra node. From that moment on ultra nodes subnet functions analogous to the pure network and routes the query according to the similarity of the known (\textit{family} and \textit{other}) ultra nodes. Besides forwarding the query to other ultra nodes ultra node may forward the query to its leaf. Every ultra node has node descriptions of its leaf nodes and if it finds that a leaf node description is similar enough to the query vector, it forwards the message to that leaf node. Leaf node further examines the query message and if the query matches any of its documents it replies directly to leaf that originated the query. In addition to response vectors \textit{QueryResponse} message carries node description of the responder's ultra node. This information will be used to cluster similar leaves together.

3.2.3 Leaf migration
Every leaf keeps track of ultra nodes and number of results it received from their leaves. If a leaf is in the right community (after a significant number of queries) the number of results it received from the best other ultra node should be comparable to the number of results it received from its own ultra node. Otherwise, if node is in the wrong community it will receive most of its replies from leaves that are not neighboring. In that case, after the leaf node has concluded that there is a better ultra node available (semantically more fitting), leaf node sends a \textit{TransferRequest} message (message 1 on Fig. 7) to the ultra node whose leaves are responsible for most results providing that it cannot be found in the negative transfer attempts cache. Every leaf nodes maintains this cache of ultra node ids that returned negative \textit{TransferReply} messages so that it wouldn't subsequently send the same \textit{TransferRequest} messages to same ultra nodes (this cache is periodically cleared).
3.2.4 Load balancing

As leaves migrate and cluster themselves some ultra nodes begin to accumulate more and more leaves (until they are full) and other ultra nodes lose leaves as they migrate to other ultra nodes. In order to balance the load (leaf connections) between ultra nodes a set of messages is defined. Following variables are defined:

- \( M \) – maximum number of leaves
- \( Lf \) – number of leaves at balance source node
- \( Lb \) – number of leaves at balance destination node
- \( p \) – percentage of leaves that balance source node gives away (we use \( p = 50\% \))
- \( T \) – threshold (if an ultra node has more or equal to \( M*T \) leaves then it may be considered as balance source node). We’ve set threshold at 70%.

When an ultra node has too few leaves (less than \( Lb \)) it sends a BalanceRequest message to a neighboring ultra node. \( Lb \) is determined from the equation (which states that balance destination node should not have more than \( T*Lf \) leaves after the balancing process):

\[
Lb = \frac{M}{T} \leq T*Lf
\]

which, if \( Lf \) is set to \( M \) as the worst case, evaluates to:

\[
Lb = M*(T-p)
\]

Therefore, when an ultra node falls down to less than \( M*(T-p) \) leaves (i.e. in our simulations less than 20% of \( M \)) it starts to send BalanceRequest messages. Related to this is the estimation of the similarity factor that a leaf node uses to determine whether to apply for transfer: if a leaf node finds that another ultra node's leaves provide more results than similarity factor times number of results its current ultra node provided, it then applies for transfer. Similarity factor \( (Sf) \) is estimated as:

\[
Sf = \frac{(Lf*p - 1)}{M}
\]

stating that a leaf that has just been balanced should be satisfied with a number of results it received from its new ultra node \( (Lb*p-1) \) even though some other full ultra node is providing more results \( (M) \). From equation (4) we get:

\[
Sf > \frac{1}{(T*p) - 1/M}
\]

On the other hand, setting \( Sf \) too high would slow up the process rendering network inert. That's why we define \( [Sf_{min}, Sf_{max}] \) interval and every node starts with \( Sf_{min} \) (making it more mobile) that is incremented on every transfer until it reached \( Sf_{max} \) (leaf nodes become less mobile as they "grow old"). We use interval \([1,7]\). Setting low starting factor produces more initial traffic but also facilitates faster node clustering.

Balance initiating node will first use family links to send a message and, if all of them fail, start to use other links collection. If an ultra node that received BalanceRequest message (marked with 1 on Fig. 8) doesn't have enough leaves than it forwards the request (message 2 on Fig. 8) using random family link (bloom filter is used). If a message finally reached ultra node that qualifies as balancing source, then a BalanceReply message is sent (message 3 on Fig. 8) to the requester with a list of nodes that can be reassigned. When the requester node receives the balance reply message it sends JoinMe messages to proposed leaves (messages 4 and 5 on Fig. 8).

Upon receiving JoinMe messages leaf nodes update their ultra node link and send LeafLeft message (messages 6 and 7 on Fig. 8) to the old ultra node. When an ultra node receives LeafLeft message, it deletes the respective link from its leaf links collection.
3.2.5 Voting process – forming family links

Ultra node forms its family links based on the votes it receives from its leaves. Leafs periodically compile a list of N ultra nodes that have been responsible for the majority of results and, if the number of results is bigger than defined minimum size (to reduce the traffic), send their votes via VoteMessage to their ultra node. Upon receiving their leaves' votes ultra nodes update their routing tables.

Aside from the voting process, hybrid network's ultra nodes subnet behaves analogous to the pure network. That is, beside family links collection it maintains other links collection and routes queries according to the same principle. Fig. 9 shows a hybrid network consisting of 3 semantically very distant communities with 2 family links, 3 other links and 10 leaves per ultra node after 100 iterations (100 queries by each node).

Fig. 9 family links + other links = all links (100 nodes)

If a leaf node fails (disconnects) in hybrid network, its ultra node simply discards the corresponding link. If an ultra node disconnects its leaves repeat the bootstrapping process and (since ultra nodes from their semantic community are probably on top of their list) hopefully find their new ultra nodes within the same community.

4 Conclusion

Content-based search is a challenging problem in P2P environments. Many researches have proposed structured P2P networks that organize routing structures according to the underlying semantic space and use some kind of distributed hash table functionality to achieve fast (and scalable) retrieval. We focus our attention on situations when there is no prior knowledge of the semantics of data. To that purpose, a set of messages and algorithms for pure and hybrid self-organizing P2P network has been proposed. In the proposed network nodes organize themselves according to the semantics of data they share: similar nodes are clustered into semantic communities. This way, most queries will not have to travel outside their communities. We find pure network simpler, more robust and less susceptible to obstruction while hybrid network reduces query network traffic and therefore scales better. Although we've performed only preliminary experiments, we believe that this approach shows promise. Network is simple and robust which is important in transient P2P environments. However, proposed network is expected to incur higher routing and network traffic cost when compared to structured P2P systems. Future work includes extensive simulations including analysis during frequent join/leave operations (for the purposes of testing we're using PlanetSim – an overlay network simulation framework [8]) and tuning of the system's performance as well as dealing with some real world aspects of the protocol like malicious peer activities and network obstruction.

References: