Semantic Overlay Networks

Arturo Crespo and Hector Garcia-Molina

Write-up by Pavel Serdyukov

Saarland University, Department of Computer Science

Saarbrücken, December 2003

Content

This paper was produced as a write-up to a presentation as part of the seminar "Peer-to-peer Systems", given by Prof. Dr. Gerhard Weikum, Chair for Database Systems and Information Retrieval, in the winter semester 2003/2004. Presentation took place at December 9th 2003. Topic of the presentation was "Semantic Overlay Network by Arturo Crespo and Hector Garica-Molina". Their paper of the same name [1] served as a basic source for my presentation and this write-up.

1 Motivation

Peer-to-peer (P2P) networks have become an important infrastructure during the last years, and P2P networks have evolved from simple systems like Napster and Gnutella to more sophisticated ones based on distributed hash tables, such as CAN and CHORD. Although, schemes, based on hash functions, provide good performance for point queries (where the search key is known exactly), they almost don't work for approximate, range, or text queries. In this case we must flood messages, like Gnutella does and loose scalability and performance.

Obviously, for such queries we have to build some different infrastructure, seemingly, based on semantic relations among peers and data, they contain. There are two main intuitions that come to mind:

- **queries can be routed only to a *semantically chosen subset of peers, able* to answer queries**. If a peer cannot answer a query fully enough, it forwards the query only to its neighbors, which can also have answers and so on. Finally, the amount of flooding messages is reduced.
- **shared data in the P2P systems often has pronounced ontological structure**, because of its origin and relations to real world concepts (music, scientific papers, and movies) and it's possible to sort such data into parts, classify its content somehow and identify semantically similar groups.

These guesses were realized in conception [1], and presented in this write-up with several extensions from other papers.

2 Introduction to Semantic Overlay Networks

Conceptual idea, suggested in [1], consists of creating a flexible network organization, improving query performance and based on the semantic relations among peers.

Main principles, incorporated in this approach are next:

- peers are clustered according to content, which they contain;
- these clusters overlap, because peers can contain different content and belong to several clusters;
- query, coming into the network, is distributed to relevant clusters only and flooded among relevant peers;
- so, clusters, irrelevant to query, don't receive any messages;

So, to get such kind of network, we have to build system of alternative networks, built on the top of the present physical network. Such network must contain only peers, relevant to each other and represent one cluster of peers.

Such networks can be called **Semantic Overlay Networks**. Each Semantic Overlay Network, or SON, represents **virtual, abstract and independent layer** of previously clustered, classified peers. Such networks play roles of mediators between queries and certain peers, they are responsible for "understanding" the meaning of query, establishing semantic relations between query and peers and implementing query routing to relevant peers and, finally, they significantly reduce overflooding of physical network

2.1 Formal definitions of SON

2.1.1 Definition, based on link structure

Semantic Overlay Network can be represented as a set of links, or triples (n_i, n_j, L), where n_i and n_j are the connected peers and L is the name of concept, in the conext of which these peers are connected.

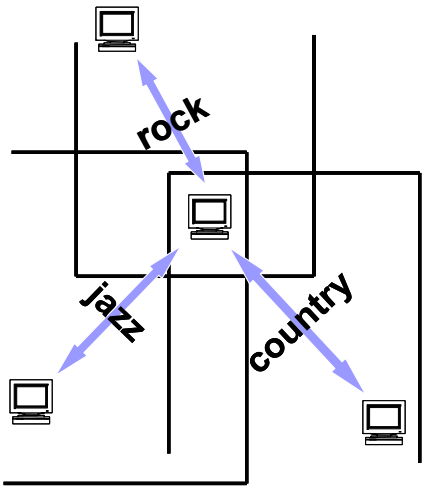

Fig. 1: Peer in the scope of different SONs

For example, at the beginning, we have only single overlay network (i.e., all links have the same L). However, we can build alternative overlay networks on the top of physical network with different L. For example, for peers, containing music, we can see at Figure 1 the following.

Peer can be connected to a set of neighbors through L = "ROCK" link in the scope of "ROCK"-SON, and to a potentially different set of peers through L = "JAZZ" or L = "COUNTRY" link in the scope of corresponding SONs. Connections can be bi-directional, as showed on figure, of one-directional (usually).

The main goal, afterwards, is to gain the best distribution of peers among SONs, which can increase performance of searching.

2.1.2 Definition, based on classification hierarchy

Also, there can be given another definition of semantic overlay network, more trivial and useful. It would be better, eventually, incorporate the concept of hierarchy into the present model, because of hierarchical nature of most of the present ontologies. ***Semantic Overlay Network*** is an overlay network, associated with a concept of a classification hierarchy. Thus, documents of the each peer must be assigned to concepts of used taxonomy, so that this peer could be assigned to corresponding SONs. For example, in Figure 2, there is shown two possible classification hierarchies for music documents. In the first one, music documents are classified according to their style and substyle; in the second one, they are

classified by tone. For example, in the leftmost hierarchy in Figure 2, we will define 9 SONs: 6 associated with the leaf nodes - **substyles**, 2 associated with intermediate nodes - **styles**, and a final one associated with root node - **music**. The example with music files will be used through all this write-up, because main experiments were carried out with Napster database and with ontology of music styles exactly.

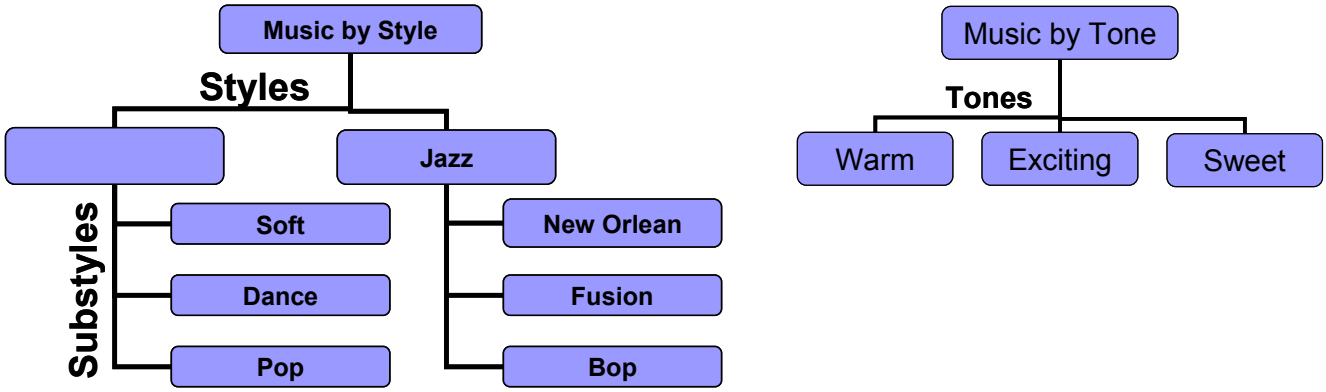

Fig. 2: Possible common-used classification hierarchies

3 Generating SONs

To allow SON's principles to start functioning, it is assumed, that it is necessary to implement 3 basic functions:

- *Join*(n_i), where one or more links of the form (n_i, n_j, l) are created;
- *Search*(q) that returns a set of peers with matches for request q;
- *Leave*(n_i) where we drop all the links in *SONs* involving n_i.

The implementation of the functions *Join*(n_i), *Search*(q) and *Leave*(n_i) will vary from system to system. Additionally, these functions may be implemented by

- each peer of the network,
- a subset of it, or even
- be provided by a computer outside the network.

Implementation of these functions is the aim of the following research, which was founded on the approach, stated here. As a whole, the process of building and using SONs is depicted at Figure 3 and consists of the following steps and stages.

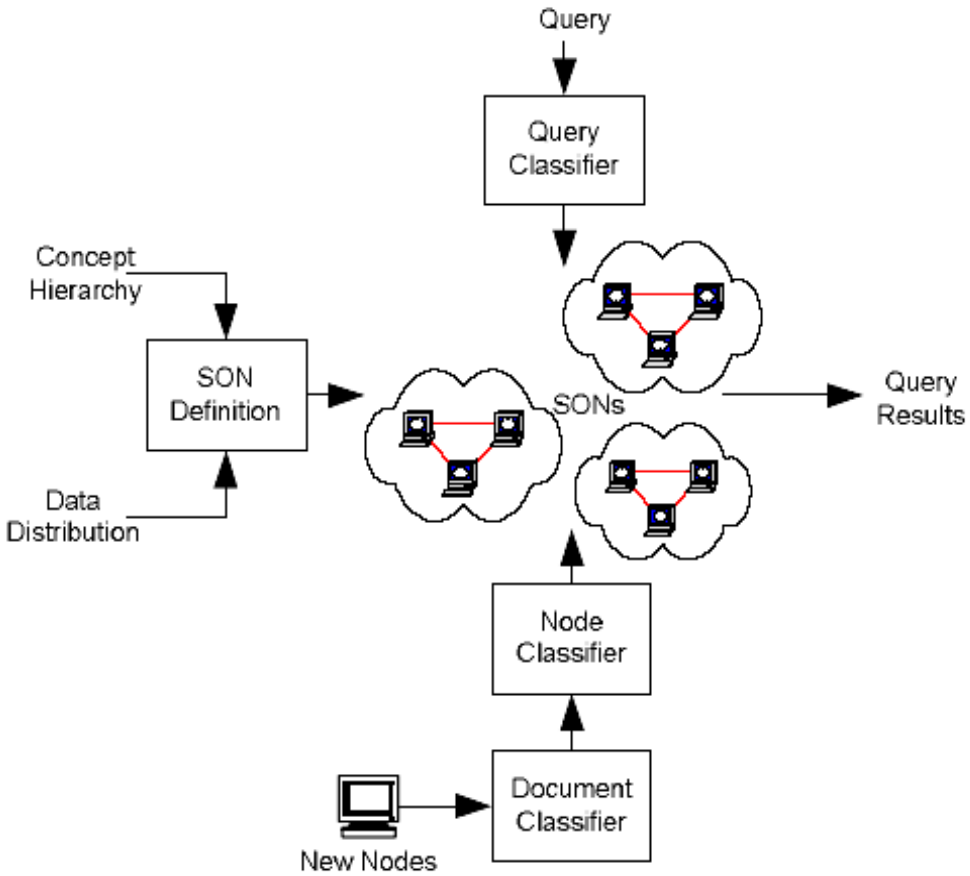

Fig. 3: Generating Semantic Overlay Networks

Preparation step:

- As it was said, at first we **evaluate and determine some classification** to be used through out all the system. This hierarchy must be stored by all of the peers in the system.

Peer joining steps:

- A peer, joining the system, first floods the network with requests for this **hierarchy** in a Gnutella fashion.
- Then, the peer runs a **document classifier** based on the obtained hierarchy **on all its documents**.
- Then, a **peer classifier** assigns the peer to specific SONs.
- The peer **joins** each SON by **finding peers** that belong to those SONs. This can be done again in a Gnutella fashion (flooding the network until peers in that SON are found) or by using a central directory.

Query answering steps:

- At first, peer issues a query (user does it);
- peer classifies query and build the list of SONs, to which query must be distributed;
- peer searches for appropriate SONs in a similar fashion as when the peer connected to its SON;
- peer sends query to the appropriate SONs.
- After the query is sent to the appropriate SONs, peers within the SON try to find matches by using some propagation mechanism (again, for example, Gnutella flooding)

Further, I will present some more sophisticated approaches of creating SONs and implementing basic functionality, stated in different papers.

3.1 Semantic routing indices

It is assumed, that every node in the scope of one SON knows only its semantic neighbors. And also, it is assumed, that, peer searches in the responsible SON in Gnutella like fashion. Of course, this is extremely inefficient even when the amount of peers is significantly reduced, using SON approach. Applying the same intuitions, it can be obvious, that peers can differ, according to relevance to this SON and to certain query. It was proposed in [2], to incorporate the neighborhood and goodness knowledge in the structure of routing indices, kept at every node, so, that to find necessary results faster.

The main principle of semantic routing indices is to keep at every node not only information about its relevant neighbors, but also the degree of relevance for every neighbor. This degree, evidently, can be expressed by the number of documents, which neighbor keeps on the category of SON in the scope of which nodes are connected. Of course, if it is necessary to gain all relevant documents, then we still need to propagate the messages through all the nodes, but if user tends to be satisfied with predefined number of results, this approach significantly improves performance.

It was also proposed in [2] the more complex approach to such routing indices: **hop-count routing indices**. To illustrate, let's see at figure 4, there is node W with this hopcount routing indices, that has three neighbors: X, Y and Z.

Node	1 Hop number of docs	2 Hops number of docs
X	13	10
Y	0	31
Z	2	10

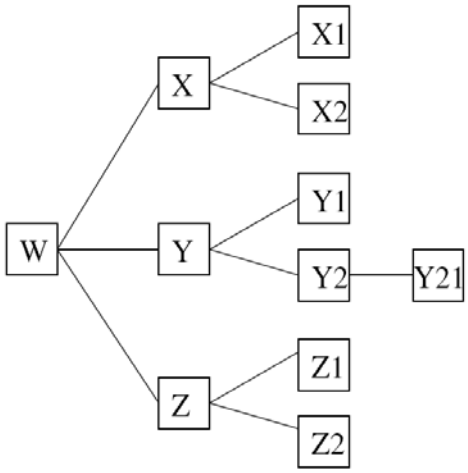

Fig. 4: Hop-count semantic routing index

With one hop via neighbor X, the node can find 13 relevant documents. The node can also find 10 more documents through X with 2 hops (i.e., at X's neighbors). Note, that we do not have information beyond the predefined horizon (number of possible hopes). So, we can define goodness of a neighbor as the ratio between the number of documents available through that neighbor and the number of messages required to get those documents. Thus, a neighbor that allows us to find 3 documents per message is better than a neighbor that allows us to find 1 document per message.

In [], they apply simple model to compute this ratio - the regular-tree cost model. The model assumes that document results are uniformly distributed across the network and that the network is a regular tree with fan-out F. Under these assumptions, it takes F^h messages to find all documents at hop h. Therefore, we can compute the number of documents per message by dividing the expected number of result documents at each hop by the number of messages, needed to find them. Formally, the ***goodness*** of $Neighbor_i$ with respect to query Q can be defined as:

$$goodness(Neighbor_i, Q) = \sum_{j=1..h} \frac{N_i[j]}{F^{j-1}} \qquad (1)$$

where h is the predefined horizon, $N_i[j]$ is the routing index entry for j hops through $Neighbor_i$. For example, if we assume F = 3, the goodness of X for a query would be 13+10/3 = 16.33 and for Y would be 0+31/3 = 10.33, so we would prefer X over Y. Of course, F can depend on each peer (number of connections to other peers).

3.2 Exploiting super-peer network

Super-peers can introduce hierarchy into the network in the form of super-peer nodes, peers which have extra capabilities and duties in the network [3]. A super-peer is a node that acts as a centralized server to a subset of clients, e.g. information provider and information consumer. Clients submit queries to their super-peer node and receive results

from it. However, superpeers are also connected to each other as peers in a pure system are, routing messages over this overlay network, and submitting and answering queries on behalf of their clients and themselves.

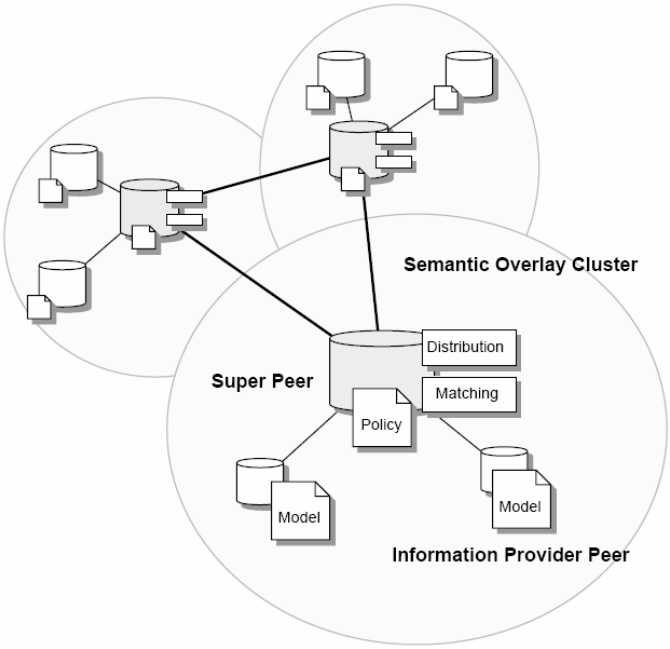

Fig. 5: Super-peer Network

Each super-peer represents a separate semantic overlay network [4], [5], [6]. Super-peers, typically computers with loads of memory and processing power, and subordinate peers, providing information, are extended with following components (see Figure 5):

- **information provider model.** The model contains a semantic rich description of the underlying peer, including information about classification aspects.
- **clustering policies** Policies describe constraints on information provider peers for each cluster. Since policies are defined by an human expert, they have to be formalized in some way, so algorithms can match suitable information provider automatically.
- **matching engines** Information provider model and clustering policies are matched against each other by a matching function. If a match occurs, a peer joins a superpeer. Matching is detected by a matching engine which implements the matching functions. Matches can either be exhaustive, partial, fuzzy or ontology based. Of course, each super-peer can also keep information about relevance of each peer to increase query search performance.
- **model distribution engine** Since each super-peer owns a separate implementation of a ”personal” matching engine and its specific super-peer dependent clustering policy, models of information provider peers, willing to join one or more superpeers, are distributed to all super-peers in the in super-peer network (as long as used classification hierarchies). This is done by a broadcast.

4 Improving SON performance

4.1 Evaluation of classification hierarchies

As it was said, classification is extremely important and, in fact, it defines the properties of SONs. It's important to distinguish good and bad classification hierarchies. The next features, or conditions, of good classification were proposed:

- **According to good classification, SONs must have connections with small number of peers, because the smaller the number of peers we need to search, the better the query performance**.

So, if some of the concepts of our ontology is really more popular than other, than almost every peer will have documents of this concept and will be connected to its concept SON. So, SON will consist of many, many peers. To illustrate, consider a classification hierarchy for a music-sharing system that it is based on the decade the music piece was originally created. In such a system, we may expect that a large number of peers will have "90's or current" music. If that is the case, there is little advantage to create a SON for "90's or current" music, as this SON will have almost all peers in the system and it will not produce any benefit benefit (but the system will still be incurring on the cost of an additional connection at each node and of having to classify nodes and queries).

- **According to good classification, peers must have connections with small number of SONs, as each connection with each SON needs to be maintained in some way by the peer. The greater the number of SONs, the greater the cost for a peer to keep track of all of them**.

So, if classification hierarchy has very high granularity, than peer seems to have many categories of documents and, therefore, be connected to many SONs. For example, consider a classification hierarchy for a music-sharing system that is based on a random hash of the music file. If we assume that nodes have a lot more files than there are hash buckets, then we can expect with a high probability that a node will have to join *all* SONs in the system. In this case, the node will have to process every single query sent into the system eliminating all the benefits of SONs.

- **it allows to use easy-to-implement classification algorithms that make a low number of errors (or no errors at all).**

To illustrate, consider an image sharing system with a classification hierarchy with the concept "has a person smiling." This concept may generate a good number of small SONs, but it requires a very sophisticated classification engine that may generate a large number of erroneous results.

4.2 Tolerance to classification errors

Indeed, there are many sources of errors when using document classifier.

- the format of the files may not follow the expected standard, so the extraction of the author and song title may return erroneous values;
- it was assumed that all files were, for example, music files for experiments with mp3 files, but, Napster, for example, allows to share other kinds of files;
- users make misspellings in the name of artist and/or song.

To evaluate the document classifier, there was measured the number of incorrect classifications. Automatic classification was compared to manual. It was considered, that classification is incorrect for a given document if the document classifier returned one or more substyles to which the document should not belong. So, experiments with Napster database showed, that 25% of the files were classified incorrectly.

Of course, these errors add additional unnecessary overhead to the system of SONs, because peers become members of improper SON's and query is distributed to additional unnecessary amount of peers and so on. But, it is more important for peer to be classified at least to proper SONs, in addition to improper ones, because serious level of data inaccessibility will make this approach almost useless. But, however, it turned out, that, a peer can **still be correctly classified even if some of its documents are misclassified because other peer's documents from the same category can be classified properly and, finally, peer can be properly classified**.

To evaluate the true effect of document misclassification, authors began to consider a classification to be incorrect for a given peer (but not to document already) if the peer was not assigned to one or more substyles to which the peer should belong. And so, from this point of view, it was found that only 4% of the peers were classified incorrectly and, thus, this approach still makes sense.

4.3 Peer assignment strategies

It was also discovered, that other features of good classification strongly depend on the strategies of peer joining.

- Most obvious strategy is the **conservative strategy,** when we place a peer in *SONc* if it has any document classified in *c*. Main drawback – peer usually has connections to many SONs and have to maintain these connections.
- A **less conservative strategy** will place a peer in *SONc only* if peer have "significant" number of document of concept *c*. In this case, we can:
  - reduce the number of peers in each SON and
  - reduce the number of SONs to which a peer belongs.

  Main drawback – we can't find documents if some relevant peers are not connected to relevant SONs.

So, final solution, which authors suggested is to use more sophisticated approach – **Layered SON's.**

5 Layered SONs

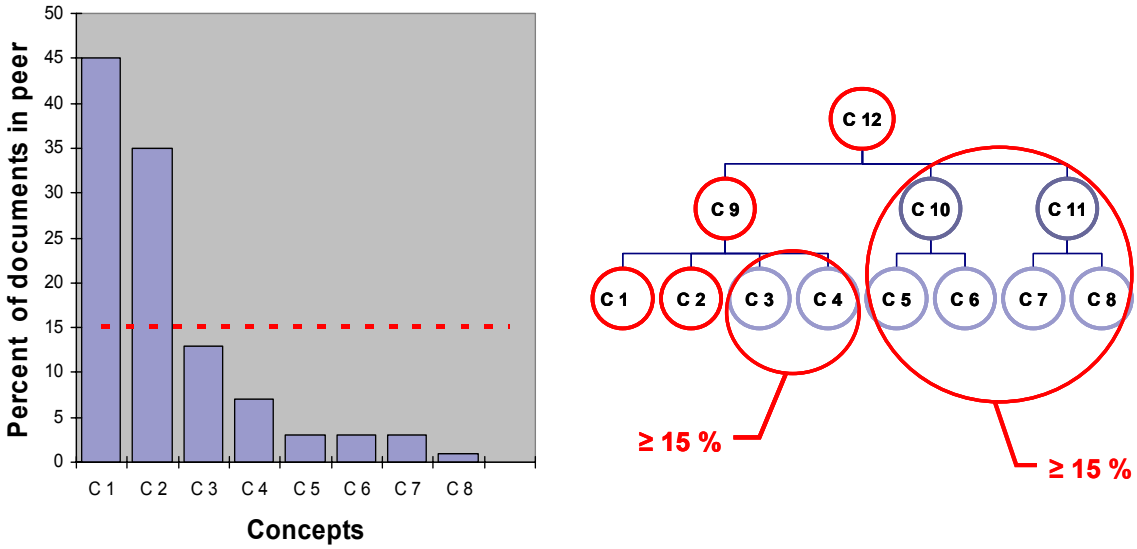

Fig. 6: Layered SONs generating stages. To the left: Apply less conservative strategy with threshold parameter; To the right: Consider combination of "non-assigned" concepts

Here (Figure 6, left), we can see the distribution of concepts in one peer. For example, peer has documents of 8 concepts. We assume, that peer can join SON of concept C, if it has at least 15% of documents of concept C, i.e we apply less conservative strategy. However, this would prevent the system from finding the documents in the node that do not belong to categories c1 and c2. How could they be found?

To let other documents of the rest of SONs to be found, we then consider the combination of "non-assigned" categories and try to join peer, at least, to "upper-level" SONs. For example, If the combination of the non-assigned nodes c3 and c4 is higher than 15%, the peer joins SON_{c9}. However, the peer does not join the SON_{c10} as the combination of c5 and c6 are not above 15%. Similarly the peer does not join the SON_{c11} as c7 and c8 are not above 15%. But, combination of c5, c6, c7, c8 is above 15%, so peer joins SON_{c12} Now, we can be sure, that all the documents can be eventually found. Let's look at searching procedure on detail.

Note, that the conservative assignment is equivalent to a Layered SON where the threshold for joining a SON has been set to 0%. In this case, the node will join the SONs associated with all the base concepts for which it has one or more documents.

5.1 Searching with Layered SONs

Searches in Layered SONs are done by first classifying the query (Figure 7a). Then, the query is sent to the SON (or SONs) associated with the base concept (or concepts) of the query classification. Finally, the query is progressively sent higher up in the hierarchy, because, upper levels still can contain relevant results and, as we saw already, nodes can join non-leaf SONs, because of lack of necessary documents. In case more than one

concept is returned by the classifier, we do a sequential search in all the concepts returned before going higher up in the hierarchy.

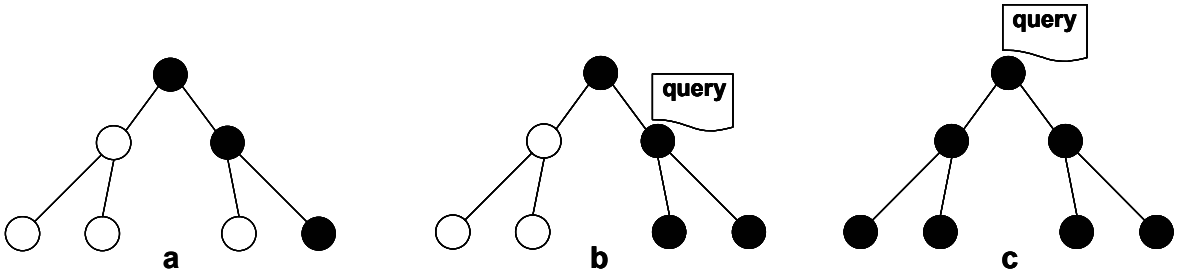

Fig. 7: Query classification examples

In practice, classification procedures may be **imprecise** and classify query at some intermediate node of ontology. In this case, we loose in performance, because we have to search more nodes at the very beginning of searching procedure (Figure 7b). For example, if we classify query at the root of ontology (Figure 7c), query results can actually be in any of the leaf concepts in the hierarchy and therefore documents classified in any category in the system can match the query. In this case, we don't receive any advantages, comparing to Gnutella-like approach, and we have to propagate query through all the peers.

So, the more precise the classification of query is, the smaller the number of concepts that need not be considered for a match. In addition, the more precise the classification of documents and peers is, the smaller the number of documents that will be classified in the intermediate nodes of the hierarchy, thus also reducing the number of documents that need to be considered for a match.

6 Comparative experiments

6.1 SONs vs. Layered SONs

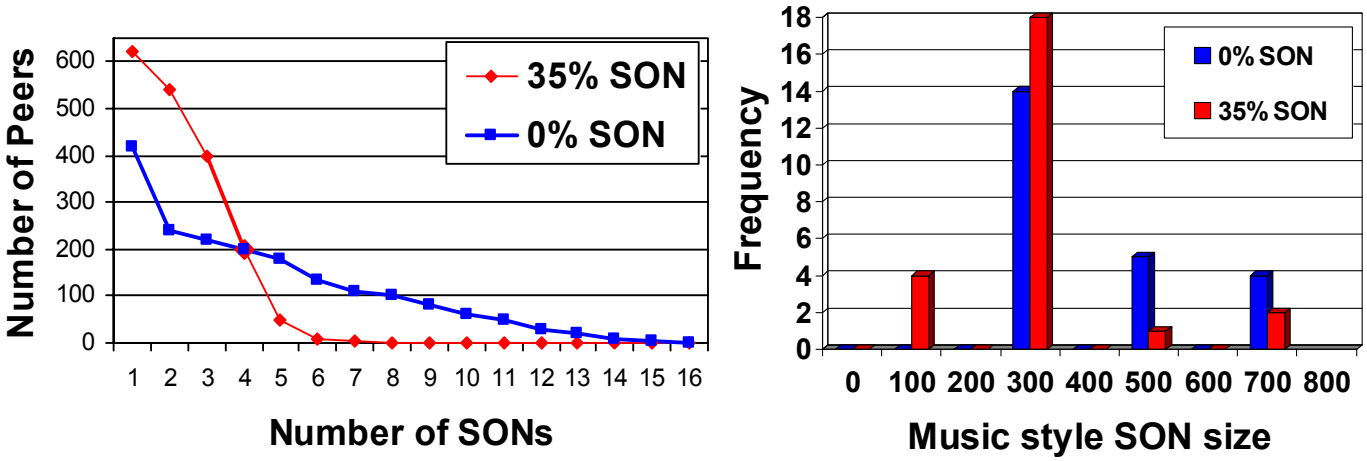

Fig. 9: Distribution of SONs and SON size

So, now, let's move to some interesting experiments, which authors produced. In left Figure, it is shown the distribution of 1800 peers among 16 music styles SONs. For Layered SONs they used threshold of 35% and conservative assignment was done with 0% threshold. We can see, from the graph, that using Layered SONs, we decrease the number of SON's with which peers need to maintain connections.

- For example, we can see that more than 600 peers (about 34% of the total peers) need to belong to just one style, when we use Layered SON approach and using the conservative approach we have only 24% of the peers, which need to belong to one style. Moreover, 97% of the peers need to belong to four or less style categories, versus 90% when doing conservative assignments. So, layered approach help peers belong to less SONs.
- Right Figure shows a histogram for the size of the SONs. From the graph we can see that by using Layered SONs we have a larger number of small SONs. Obviously, this reduction will lead to significant improvements in query performance. So, using layered SONs also helps reduce the number of peers per SON.

Authors had also done their experiments with SONs for leaf nodes of classification hierarchy - for music substyles. In this case, layered approach doesn't give much in the sense of reducing number of connections between peers and SONs. But, it still significantly reduces the size of SON's.

6.2 Layered SONs vs. Gnutella

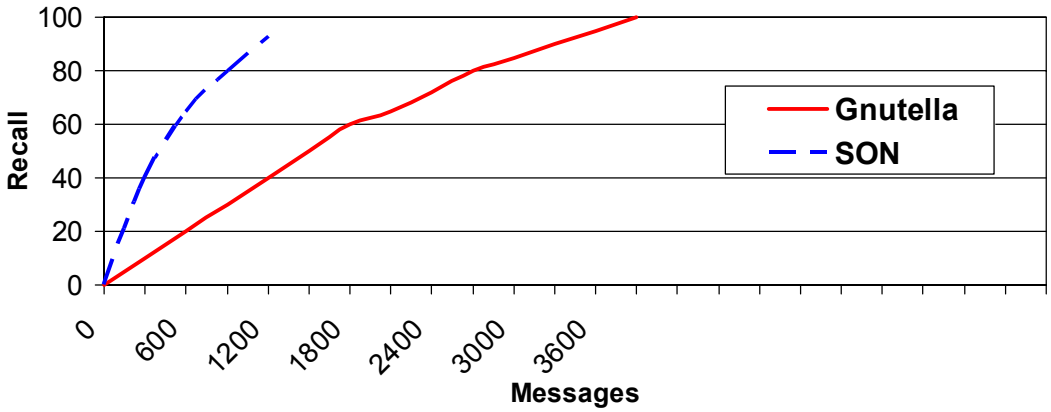

Fig. 10:Number of messages for one query

At the chart, we can see, that Layered SON system were able to obtain the same level of matches with significantly fewer messages than the Gnutella-like system. And this is the main achievement of SON approach, indeed. Nevertheless, there are few drawbacks, restricting usage of SONs in some way:

- At first, it must be realized, that search algorithm, based on SON's **can't reach the 100% recall level.** If we to find all documents for a query, our only option is an exhaustive search among all peers in the network. However, when we have **25% of misclassified documents,** it is still possible to find **more than 93% of the documents** that match a query. So, this is excellent **recall level**, nevertheless
- And, this search algorithm may result in duplicate results. Specifically, duplication can happen when a peer belongs, at the same time, to a SON associated with a **substyle** and to the SON, associated with the **parent style of that substyle** (peer can have some, but not many, documents on some other substyles of this style). In this case, a query that is sent to both SONs will search the peer twice and thus it will find duplicate results. But, obviously, technically this is solvable problem.

7 Summary

Presented papers show how to improve the efficiency of a peer-to-peer system by clustering nodes with similar content in Semantic Overlay Networks (SONs). SONs can efficiently process queries while preserving a high degree of node autonomy. There were also introduced Layered SONs, an approach that improves query performance even more, at a cost of a slight reduction in the maximum achievable recall level. From experiments, it's can be seen, that SONs offer significant improvements versus random overlay networks, while keeping costs low. Also, there are other contributions. It was shown well, that the super-peer topology, consisting of a super-peer backbone with powerful computers and smaller clients which are linked to these super-peers, is very suitable for this approach. Also, the method of using semantic routing indices for robust search within clusters was presented. Tracing last publications on this topic, it is possible to conclude, that these two sub-researches are most actual among all attempts of developing presented ideas.

8 Literature

[1] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks. 2003. Submitted for publication.
http://www-db.stanford.edu/peers/
Extended version, 2002:
http://citeseer.nj.nec.com/garcia02semantic.html

[2] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. Proceedings of the International Conference on Distributed Computing Systems (ICDCS). 2002.

[3] B. Yang and H. Garcia-Molina. Designing a super-peer network. In Proccedings of the ICDE, March 2003.

[4] W. Nejdl, M. Wolpers, W. Siberski, A. Loser, I. Bruckhorst, M. Schlosser, and C. Schmitz. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In Proceedings of the Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.

[5] A. Loser, F. Naumann, W. Siberski, W. Nejdl, U. Thaden. Semantic Overlay Clusters within Super-Peer Networks. Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB. 2003.

[6] A. Loser, M. Wolpers. Efficient data store and discovery in a scientific P2P network. Technische Universitat Berlin, Learning Lab Lower Saxony Hannover. Submitted for publication. 2003.