P2Prec: a Social-based P2P Recommendation System for Large-scale Data Sharing

Fady Draidi, Esther Pacitti, Patrick Valduriez, Bettina Kemme

P2Prec: a Social-based P2P Recommendation System for Large-scale Data Sharing

Fady Draidi, Esther Pacitti, Patrick Valduriez, Bettina Kemme

INRIA & LIRMM, Montpellier, France
{*Fady.Draidi@lirmm.fr, Esther.Pacitti@lirmm.fr, Patrick.Valduriez@inria.fr*}

McGill University, Montreal, Canada
{*kemme@mcgill.ca*}

Abstract. We propose P2Prec, a P2P recommendation system for large-scale data sharing, which exploits friendship links. The main idea is to recommend high quality contents related to query topics and contents of friends (or friends of friends), who are expert on the topics related to the query. Expertise is implicitly deduced based on the contents stored by a user. To exploit friendship links, we rely on Friend-Of-A-Friend (FOAF) descriptions. To disseminate information about experts, we propose new semantic-based gossip algorithms that provide scalability, robustness, simplicity and load balancing. By using information retrieval techniques, we propose an efficient query routing algorithm that recommends the best peers to serve a query. In our experimental evaluation, using the TREC09 dataset and Wiki vote social network, we show that using semantic gossiping increases recall by a factor of 2.5 compared with well known random gossiping. Furthermore, P2Prec has the ability to get reasonable recall with acceptable query processing load and network traffic.

Keywords: P2P systems, social-based recommendation, gossip algorithms, semantic-based gossiping, information retrieval.

1 Introduction

Collaborative web 2.0 tools such as social networks, wikis, and content sharing web sites make it now very easy to publish and share huge amounts of data, content and knowledge, among very high numbers of users over the network. Similarly, in modern e-science (e.g., bio-informatics, physics and environmental science), scientists must deal with overwhelming amounts of experimental data produced through empirical observation and simulation. Such data must be processed in a collaborative way among different researchers, perhaps from different laboratories, in order to draw new conclusions, produce knowledge or prove scientific theories. Scientists typically work and collaborate using complex workflows that involve hundreds or thousands of processing steps, access terabytes of data, and generate terabytes of result data. With the constant progress in collaborative tools, scientific observational instruments and simulation tools, the data overload keeps worsening and makes centralized data sharing difficult.

Peer-to-Peer (P2P) networks, offering scalability, dynamicity, autonomy and decentralized control, can be useful for large-scale data sharing. So far, P2P has been primarily used for content-sharing, examples of popular systems being BitTorrent [4] and eMule [9]. Recently, P2P has also been applied to support high performance scientific workflow computing [20]. The popularity of P2P systems has translated into huge amounts of data being spread over increasingly larger number of peers (and users). As more data becomes available, users tend to get overwhelmed with the high numbers of documents returned as results of their queries, and it becomes hard for them to find the most valuable and relevant documents. In addition, popular P2P content-sharing systems such as eMule only provide a very simple keyword search capability, trying to find the documents whose name or description match the keywords provided by the user. The same observation can be made in scientific applications. Consider the typical case (e.g., in biology) where experimental data sets are stored in raw format and their contents are described in associated documents (i.e., published scientific papers). When a scientist needs to select a data set that best matches her requirements for a workflow execution (i.e., scientific question), she needs to understand the candidate raw data, using the associated documents. In this case, the challenge is to find those documents from a very large collection, that are most relevant to the scientific question.

The general problem with current P2P content-sharing systems is that the users themselves, i.e., their interest or expertise in specific topics, or their rankings of documents they have read, are simply ignored. In other words, what is missing so far is a recommendation service, also called recommender system or denoted as RS, that can recommend high quality and valuable documents exploiting user information. Recommendation is ubiquitous in our daily life, where we must choose between alternatives based on opinions and advices that we have received from other resources such as people we know (friends, family members, etc.), experts we trust, general surveys, travel guides, published reviews, etc. In order to enable people to share their opinions and advices, and benefit from each other's experience without human intervention, RSs have emerged. RS exploit the users' social data (interest, expertise, friends, etc.), and suggest documents or information items (e.g., movies, documents, Web pages, CDs, or books) of interest to users according to their interests [11]. However, most of the existing RSs follow a centralized architecture or do not exploit the users' social data [1] as we do.

In this paper, we propose *P2Prec*, a P2P RS for large-scale data sharing that exploits the users' social data. The primary applications of P2Prec are P2P content-sharing and scientific data sharing. To manage the users' social data, we rely on the Friend-Of-A-Friend (FOAF) project [30]. FOAF provides an open, detailed description of profiles of users and the relationships between them using a machine-readable syntax. Whenever a user (or "software on the behalf of the user") generates its FOAF file, it can obtain an identity for that file on the Web in the form of a URI. This URI could point to a reference in the user's FOAF file stored in a server that the user trusts. Thus, FOAF can be an important tool to provide simple directory services and one can use information from FOAF files to locate people. One can imagine FOAF as a way of describing a distributed directed graph of friendship relations,

where each user may specify its profile with information such as: interests, topics of expertise and friends in its FOAF file, and then stores it in a server that its trusts.

For the sake of scalability and decentralization, we choose to implement P2Prec over an unstructured P2P overlay, in which each peer represents a user. Each user may store in its FOAF file its profile with information such as topics of interest and direct friends. Among other information, each peer derives implicitly whether it is expert on specific topics. To disseminate user information about experts between friends and friends of friends, we choose to use gossiping as it exhibits important properties such as scalability, robustness, simplicity and load balancing. Applying one of the well-known random gossip algorithms [10, 15] in P2Prec, each user keeps locally a view of its friends, and friends of friends, and their corresponding topics of expertise and interests. Periodically, each user chooses randomly, or taking into account the age of the entries in its view a contact to gossip with. The two then exchange a subset of each other's view, and update their view state. This allows peers to get to know new peers and to forget about peers that have left P2Prec. Whenever a user submits a query, the view is used as a directory to redirect the query to the appropriate peers. Thus, overlay maintenance and information dissemination are done gracefully, assuring load balancing and scalability.

With random gossiping, several algorithm parameters, such as the user with whom to exchange the view, the view subset, etc. are chosen randomly. In P2Prec, users search for documents that are related to topics of interests. Thus, semantic information such as a user's topics of interests and expertise used for recommendation must be taken into account while gossiping in order to increase the quality and the efficiency of query responses. Intuitively, the choice of the user with whom to gossip should be based on user affinity in terms of expertise and topics of interest. Similarly, when selecting a view subset to exchange, the same parameters should be taken into account without hurting the properties of gossiping. In P2Prec, gossiping introduces implicit recommendation.

Gossip algorithms were initially proposed for network monitoring. With the challenges brought by distributed and large-scale data management in different domains, gossiping has been used for further purposes. For instance, in [2], the authors propose to introduce semantic parameters while gossiping for large-scale social network exchanges. However, it does not address recommendation in P2P content-sharing as we propose in this paper. In this paper, we make four main contributions.

1. We propose a P2P RS that uses social data to establish links among friends and friends of friends, in order to improve the search for relevant content. We adapt some information retrieval techniques to help P2Prec retrieve relevant documents, using the topics related to a query and to documents, and user expertise.
2. We propose two new semantic-based gossip algorithms that take into account semantic information such as the users' topics of interests and expertise, without hampering the nice properties of gossiping. These algorithms introduce the concept of implicit recommendation.
3. We propose an efficient query routing algorithm that takes into account the view content and recommends the best peers to serve a query. In addition, we propose a

novel method to rank the returned documents, by combining a document's popularity with its semantics.
4. We provide an experimental evaluation using real data sets that demonstrates the efficiency of P2Prec over the TREC09 [25] and Wiki vote social networks [33].

The rest of this paper is organized as follows. Section 2 introduces background concepts. Section 3 provides an overview of P2Prec. Section 4 describes initialization in P2Prec. Sections 5-7 describe random, semantic and semantic two-layered gossip algorithms, respectively. Section 8 describes our solution for query routing and result ranking. Section 9 gives an experimental evaluation. Section 10 discusses related work. Section 11 concludes.

2 Background

P2Prec uses LDA for automatic topic extraction, FOAF files to manage users' social profiles, and unstructured P2P networks for communication.

2.1 LDA Topic Extraction

We need a technique to extract and classify the hidden topics available in the documents that will be used to define the users' topics of expertise. Classifying the hidden topics available in a set of documents is an interesting problem by itself. Several models have been proposed, described and analyzed in the Information Retrieval (IR) literature [7] to tackle this problem. The one we use is Latent Dirichlet Allocation (LDA) [5]. LDA is a topic classifier model that represents each document as a mixture of various topics and models each topic as a probability distribution over the set of words in the document. For example, a document talking about *vegetarian cuisine* is likely to be generated from the mixture of words from the topics *food* and *cooking*.

We now explain how we adapt LDA to P2Prec where LDA processing is done in two steps: the training (at a global level, see interface 1 in Figure 1(a)), and inference (at the local level, see interface 2 in Figure 1(b)). Training is usually done by a specific peer, e.g., the bootstrap server. LDA is fed with a sample set of M documents that have been aggregated from the system, i.e., collected from P2Prec participant peers on demand. Each document $doc \in M$ is a series of words, $doc=\{word_1,...,word_n\}$, where $word_i$ is the i^{th} word in doc and n is the total number of words in doc. Then, LDA executes its topic classifier program outputs a set $B=\{b_1,.. b_d\}$ of bags (in fact a bag is a set). Each bag $b \in B$ is tagged with a label t (we refer to it as topic t in P2Prec context). The domain of topics T of P2Prec corresponds to $t_1...t_d$. Each bag contains a set of z words, where z is the total number of the unique words in M, and each of these words is associated with a weight value between 0 and 1. More formally, this set of bags can be represented as a matrix ϕ with dimensions d*z, where d is the number of topics and z is the total number of unique words in M. Each row of ϕ represents the probability distribution of a topic $t \in T$ over all words. The bootstrap server

periodically aggregates *M* from the P2Prec participants and estimates ϕ. Each version of ϕ is attached with a timestamp value.

The inference part of LDA is performed locally at each P2Prec participant user *u*. The goal is to extract the topics of *u's* local documents, using the same set of topics that were previously generated at the global level. Thus whenever a peer joins P2Prec, it first contacts the bootstrap server in order to download ϕ. Then for inference, LDA's input is the set of local documents of user *u*, and the matrix ϕ generated at the global level. As output LDA provides a vector of size *d* for each document *doc*, called document topic vector, $V_{doc}=[w_{doc}^{t_1}....w_{doc}^{t_d}]$, where $w_{doc}^{t_i}$ is the weight of each topic $t \in T$ with respect to *doc*. The detail of how LDA is used locally is presented in section 4.2.

Interface 1- Global-Training-LDA at bootstrap server
Input: Set M *docs* aggregated from P2Prec users
Output: Matrix ϕ, row_i of ϕ is a bag-of-words,
Each row_i corresponds to a topic $t \in T$
(a)
Interface 2- Local-Inference-LDA at user u
Input: T and u's local *docs*
Output: V_{doc}, for each *doc*
(b)

Fig. 1. The Interfaces of LDA under P2Prec context

2.2 FOAF Files

FOAF [30] provides a simple, machine-readable vocabulary serialized in RDF/XML to describe people, content objects and the connections that bind them all together. A FOAF file is typically created by the individual user and published on a server that the user trusts. Over the last few years, FOAF has become increasingly popular and used in many different projects [18].

With a FOAF file, a user can describe herself using the foaf:Person class, listing attributes such as name, address and expertise and use foaf:knows to describe its friends, etc. Whenever a user generates its FOAF file, it stores it in a host server that it trusts and obtains an identity for the file on the Web in the form of a URI from that host server. Overall, the FOAF vocabulary is simple and can be integrated with any other semantic Web vocabularies.

Figure 2 shows the FOAF file adapted to P2Prec. The FOAF file owner Jean includes Jean's personal information and information about her friends. In the personal information, the FOAF file shows her name and information about her topics of expertise. It shows that she is expert in topics $t_1 \in T$ and $t_2 \in T$ where t_1 and t_2 have been extracted from the documents she maintains by using the two steps of LDA. In Friends information, Jean's FOAF file shows that she knows a friend whose name is Peter, the URI of its FOAF file is http://www.lirmm.fr/Peter.rdf. The attributes degree and trust are motivated in section 3.

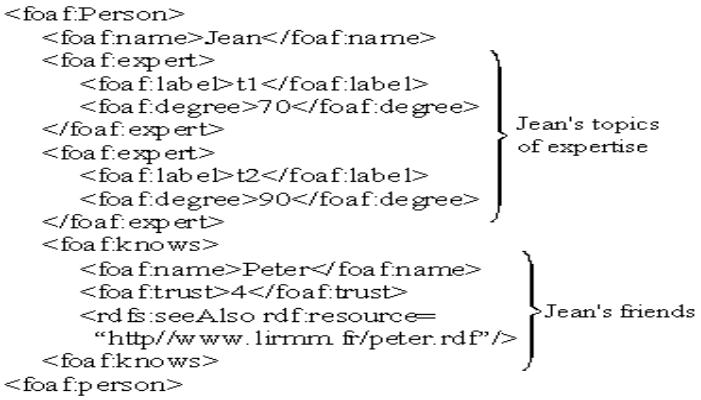

Fig. 2. An example of a FOAF file in P2Prec

2.3 P2P Networks

P2P networks can be classified according to their overlay topology between *unstructured* and *structured*. Typically they differ on the constraints imposed on how users are organized and where shared contents are placed [24]. In P2Prec design we choose an unstructured overlay because: unstructured networks impose few constraints on users' neighborhood and content placement [24] so that users in the overlay get loosely connected. This makes joining and leaving an unstructured overlay easier and results less overhead, but makes lookups a bit more complicated. Unstructured networks typically use flooding [24], gossiping [10] or random walk [24] algorithms to disseminate discovery messages or queries. With flooding, a user sends a query to all its neighbors.

Gossip algorithms [10, 15] have attracted a lot interest for building and managing unstructured networks. With gossip, each user periodically exchanges its state (a user's state might be its shared date or documents, a set of other contacts, etc.) called view, with another randomly-selected user. Thus, after a while, as with gossiping in real life, each user will have a partial view of what other users in the system know and uses it to serve its queries.

3 Overview of P2Prec

In this section, we give a basic overview of P2Prec, with the main terms and assumptions used in the paper, and introduce our query routing solution.

3.1 Basic overview

P2Prec'general goal is to improve the quality and efficiency of query responses in P2P content sharing systems, by exploring the synergy between RSs and the social relations among users. Sinha et al. [29] have shown that users prefer the advices that

come from known friends (friends, family members, colleagues) in terms of quality, confidence and usefulness. The basic idea of P2Prec is to use an adapted gossip algorithm to spread recommendation of expert users and there topics of expertise to improve query response quality once a query is submitted.

We model a P2P content sharing system as a graph $G = (D,U,E,T)$, where D is the set of shared documents, U is the set of users in the system, E is the set of edges between the users such that there is an edge $e(u,v)$ if users u and v are friends, and T is the set of users' topics of expertise. Each user $u \in U$ is associated with a set of topics of expertise $T_u \subset T$, so $t \in T_u$ indicates a topic t for which user u is an expert. The cardinality of U is denoted by $|U|$ and the cardinality of T is denoted by $|T|$.

We assume that each user $u \in U$ stores and maintains locally (on its peer) a set $D_u \subset D$ of documents that it has rated, each rate over a document $doc \in D_u$ is denoted by $rate^u_{doc}$. The cardinality of D_u is denoted by $|D_u|$. Notice that the user's rate over a document $doc \in D_u$ can be either explicit or implicit [26]. The system may ask the user to explicitly give a numeric rate for *doc*. On the other hand, the user's rate over $doc \in D_u$ may be extracted by monitoring implicitly its behavior over *doc*, e.g., the time the user spends in reading *doc*, how many times the user browses *doc*, etc. The user ratings either explicit or implicit are often represented by discrete values within a certain range, e.g., between 1 and 5.

Each user $u \in U$ also stores and maintains locally (on its peer) a FOAF file which contains a description of its personal data such as its personal information and friends as depicted in Figure 2. Recall that personal information includes u's topics of expertise $T_u \subset T$. Notice that each topic of expertise $t \in T_u$ that has been included in the FOAF file is associated with a *degree*. The degree of a topic of expertise $t \in T_u$ represents how many documents user u has in topic t. In the example of Figure 2, Jean is expert in topic t_1 and t_2. The degree of its topics of expertise t_1 is 70 and t_2 is 90 i.e., Jean has 70 documents in topic t_1 and 90 documents in topic t_2

Furthermore, user u's FOAF file includes information about its friends denoted by $friends(u)=\{f_1, f_2, \ldots\ldots f_n\}$, where n is the number of friends of user u. Friends' information includes friends' name, links (URI) to its FOAF files and trust levels. Trust level between user u and a friend v is a number vary in a range of [0,5] and it represents how much user u faith in its friend v. Trust level between user u and its friend v can be obtained explicitly or implicitly[17]. In the example of Figure 2, Jean has trusted her friend Peter with a level of 4.

For privacy issues we assume that each user can add a rule of access for each document $doc \in D_u$ it maintains. Thus, we distinguish four types of documents based on the rules that have been given by user u over a $doc \in D_u$:

1. Personal document: document that cannot be accessed by any user.
2. Confidential document: a document that can be accessed by a user or a set of users that have been chosen by user u.
3. Private document: a document that can be accessed by any honest users. A user v is considered honest user with respect to user u if it has a trust level with user u greater than a minimal trust value (system-defined), and the shortest path (number of acquaintance) between user v and user u less than a maximal distant(system-

defined). Trust level between a user u and indirect friend v can be computed by multiplying the trust levels of the acquaintances between user u and v [17].

4. Public document: a document that can be accessed by any user in the system.

We now present how concept of expert. Recall that P2Prec lets each user extract locally its topics of expertise from the documents it maintains. If a user $u \in U$ has rated at least a number x of documents (where x is system defined) in D_u, i.e., $|Du| \geq x$, then user u may become expert in specific topics $T_u \subset T$, called *topics of expertise* of u (see more details in Section 4.2). Once the user u has extracted its topics of expertise T_u, it records its T_u in its FOAF file along with their degrees of expertise (represents how many documents user u has in topic t).

The idea behind P2Prec is to let each user periodically exchange (gossip) along the system graph G, its topics of expertise (if it has) with its direct and indirect friends. Thus, each user continuously maintains a partial view of the topics of expertise of the users in the system. Consequently, a user u posing or receiving a keyword query q, uses its view to find potential expert users that might have high quality documents related to q. Notice that LDA is used to extract the topics from the query q keywords.

We assume that friends have the ability to exchange their FOAF files. Thus, each user $u \in U$ join P2Prec either expert or non expert sends its FOAF file to its direct friends and retrieves their FOAF files. As a consequence, user u's friends know about its existence and they include u in their views as well user u knows the existence of its friends and their topics of expertise and adds them to its view. As a result, the user u initializes its partial view denoted by *local-view* (more details are given in section 4.3).

We propose two new gossip algorithms: 1) semantic gossiping, that lets u selectively aggregate high and good interesting users in it view. 2) Semantic two-layered gossiping: random and semantic gossips are combined, i.e. u has a view for each algorithm. Random gossip is used to insure that new users are always taken into account in u's views. Semantic gossip is used to let u selectively aggregate high and good interesting users in it view, however taking into account the random view.

Even though P2Prec is built by gossiping between friends (of friends), users with no friends, which we call *isolated-users*, still have the ability to use the system to get high quality recommendations (see Section 7).

3.2 Query Processing

P2Prec participant users submit keyword queries. A query is defined as $q(word_i, \text{TTL}, V_q, T_q, u)$, where $word_i$ is a list of keywords, TTL is the time-to-live value, V_q is the query q's topic vector. Notice that the query q's topic vector V_q is computed by using the **Local-Inference-LDA**. T_q is the query q's topics and u corresponds to the address of the query q initiator. Query processing at each user u is illustrated in Algorithm 1. The active behavior describes how a user u initiates a query q, while the passive behavior shows how the user u reacts to a query q initiated by some other user v.

The active behavior is executed when a user u initiates a query q. Once user u initiates a query q, it routes q as follows: first, it extracts the query q topic vector V_q by using the **Local-Inference-LDA** (line 1). Then user u computes the query q's

topics T_q from q's topic vector V_q using **ComputeQueryTopics()** method (line 2). After that user u uses its *local-view* to find potential expert users that might have high quality documents related to q' topics T_q and then floods the q to them after reducing TTL by one by using **Route-query** (line 3).

In turn, passive behavior is executed whenever a user u receives a query q that has been initiated by a user v. User u that has received a query q returns to q initiator the documents it has which are related to q taking into account the rule of access that has been assigned by user u to each $doc \in D_u$, and selects from its *local-view* the users which are expert in query q' topics T_q and floods the query q to them while the query TTL does not reach zero by using **Process-query** (line 3). **Route-query (), Process-query (),** and **ComputeQueryTopics()** will be presented in detail in section 8.

Algorithm 1- Query processing at user u
//Active behavior: user u initiates a query q
Input: q ($word_i$, TTL, V_q, T_q,u); set of topics T; $local\text{-}view_u$
Output: u sends q to potential experts
1 V_q = **Local-Inference-LDA**(q,T)
2 T_q = **ComputeQueryTopics**(V_q)
3 **Route-query**(q, *local-view*)
//Passive behavior: user u receives a query q initiated by user v
Input: query q ($word_i$, TTL, V_q, T_q,u); D_u, user u's documents ; user u's *local-view*
Output: answer, a set of *docs* related to q; u sends q to potential experts
1 **WaitQuery**( )
2 **Receive** *query q*
3 *Answer* = **Process-query**(q, D_u, $local\text{-}view_u$)

4 Initialization in P2Prec

In this section, we show how users should initialize their participation in P2Prec, which include extracting users' topics of expertise and initializing users' *local-views*. Then, we describe how users extract their topics of expertise from the documents they have rated. Finally, we explain how users initialize their *local-views* when joining.

4.1 System Initialization

Algorithm 2 illustrates how a user u initializes its participation at P2Prec. Whenever user u joins P2Prec, first, it gets the set of topics T (the bags-of-words) from the bootstrap server using the **GetTrainingTopics( )** method . Recall that the set T is the set of topics that has been extracted by using the **Global-Training-LDA** on the bootstrap server. Getting the set T from the bootstrap server usually happened at the first time that the user u has participated to P2Prec or if it receives and advertisement from the bootstrap server that there is a new copy of T. After that, user u checks its possibility to become an expert user as following: first, it counts locally how many documents it rates and maintains $|D_u|$. If $|D_u|$ has exceeded a specific number x, it extracts the documents topic vectors of D_u using the **Local-Inference-LDA**. Then

user *u* computes its topics of expertise T_u using the **Compute-Topics-Of-Expertise** (line 4). If user *u* becomes an expert, it adds its topics of expertise T_u along with their *degrees* to its FOAF file using the **UpdateFOAF()** method. Then user *u* initializes its *local-view* by exchanging with its friends their FOAF files using the **InitializeView** (line 9). Finally, user *u* exchanges its *local-view* with its friends and indirect friends using the **gossiping** (line10).

Algorithm 2- **Initialization**
Input: D_u set of documents user *u* maintains; $rate_{doc}$, rates that have been given by user *u* over D_u; user *u*'s FOAF file
Output: user *u*'s starts gossiping
1 T = **GetTrainingTopics**( ) from bootstrap server
2 **If** $|D_u| \geq x$ **then**
3 V_{doc} = **Local-Inference-LDA**(T, D_u)
4 T_u = **Compute-Topics-Of-Expertise**(V_{doc}, $rate_{doc}$)
5 **If** T_u is not equal to empty **then**
6 **UpdateFOAF**(T_u)
7 **End if**
8 **End if**
9 User u's *local-view* = **InitializeView**($FOAF_u$)
10 Trigger **gossiping**(*local-view*$_u$)

4.2 Extracting Users' Topics of Expertise

Algorithm 3 illustrates how each user computes its topics of expertise. Each user *u* which has $|D_u| > x$ locally computes its topics of expertise $T_u \subset T$, in two steps. First, it computes the document quality for each document $doc \in D_u$ it has rated and records it locally in a vector denoted by *quality(doc,u).* This is done by multiplying the document topic vector $V_{doc}=[w_{doc}^{t_1} \ldots w_{doc}^{t_d}]$ that has been extracted using **Local-Inference-LDA** interface, by the rate $rate_{doc}^{u}$ that has been given by user *u* over *doc*. Thus, we have: *quality(doc,u)* = $[w_{doc}^{t_1} * rate_{doc}^{u} \ldots\ldots w_{doc}^{t_d} * rate_{doc}^{u}]$ (corresponds to line 2). Then, user *u* extracts for each topic $t \in T$ only the documents that have high quality in that topic *t.* A document *doc* is considered a high quality document in a topic *t*, denoted by $quality_t(doc, u)$, if its weight in that topic w_{doc}^{t} multiplied by its rate $rate_{doc}^{u}$ exceeds a threshold value (which is system defined), i.e.,

$$quality_t(doc, u) = \begin{cases} 1, & w_{doc}^{t} * rate_{doc}^{u} \geq threshold \\ 0, & otherwise \end{cases}$$

In the second step (lines 3, 4 and 5), user *u* counts how many high quality documents it has in each topic $t \in T$. The number of high quality documents that belongs to a topic $t \in T$ represents *u*'s degree of expertise in that topic *t*, denoted by $degree_u^t$, i.e.,

$$degree_u^t = \sum_{i=1}^{|D_u|} quality_t(doc_i, u)$$

Then user u computes its topics of expertise $T_u \subset T$ (lines 9, 10 1nd 11). User u is considered an expert in topic $t \in T_u$ if a percentage y (where y is system-defined) of its documents D_u have high quality in that topic t, i.e.,

$$\left\{ \frac{degree_u^t}{|D_u|} \geq y \right\}$$

We can use absolute values instead of percentage y, for instance we use the percentage y. Finally, u records its topics of expertise $T_u \subset T$ along with their *degrees* in its FOAF file.

Algorithm 3- **Compute-Topics-Of-Expertise**(V_{doc}, $rate_{doc}$)
Input: user u's document topic vectors, V_{doc} where $doc \in D_u$; user u's document rates, $rate_{doc}{}^u$ where $doc \in D_u$
Output: user u's topics of expertise T_u if user u becomes an expert

```
1  For each doc ∈ D_u do
2     quality(doc,u) = Multiply(V_doc, rate_doc^u)
3     For each t ∈ T do
4        If quality_t(doc,u) then
5           increase degree_u^t by one
6        End If
7     End For
8  End For
9  For each t ∈ T do
10    If (degree_u^t / |D_u|) ≥ y then
11       add t to T_u
12    End If
13 End For
```

User u has the ability to download and rate the document recommendations it receives, and add or delete documents. Thus, its topics of expertise might be changed. To capture this dynamic behavior, user u computes its topics of expertise T_u at every fixed period of time, or if a number of documents have been added to (or deleted from) its D_u and exceeds a system-defined threshold.

4.3 Initializing Users' Local-Views

Recall that the idea behind P2Prec is to let each user periodically exchange, its topics of expertise (if it has) with its direct and indirect friends. To achieve that each user u maintains a *local-view*, which contains a fix number of entries, noted *view-size*, each entry refers to a user (a user may be a direct or an indirect friend). Each entry contains the IP address of the user and user topics of expertise along with their degrees.

We limit *local-views* to a limited size *view-size* to prevent them from increasing linearly with the network size. Hence increasing *local-views* size induces scalability problem and increases the cost of maintaining their entries up-to-date.

Algorithm 4 describes how a user u initializes (fills) its initial *local-view* during the joining process. Recall that direct friends have the ability to exchange their FOAF files. Thus, each user $u \in U$ join P2Prec either expert or non expert sends its FOAF file

to its direct friends *friends(u)* (corresponds to lines 1 and 2). In turn, each direct friend $v \in friends(u)$ receives user u's FOAF, it returns to user u its own FOAF file. Then, v extracts user u's topics of expertise T_u from u's FOAF file (if u is expert), and adds user u to its *local-view*, if the size of its *local-view* is less than *view-size*. When the size of v's *local-view* become equal to *view-size*, v selects randomly a user x from its *local-view* and replaces x by u.

On the other hand, when user u receives the FOAF files of its direct friends $v \in friends(u)$ (line 3). For each friend v, u extracts v topics of expertise from v FOAF file (if v is expert). Afterwards, u adds v to its *local-view* while $|local\text{-}view_u| < view\text{-}size$ (corresponds to lines 5 and 6). Once u's *local-view* size become equal to *view-size*, u selects randomly a user x from its *local-view* using the **selectUser**() method, and replaces x by v (lines 7-9). As a result, the user u initializes its *local-view*. The initial *local-view* contains at first the entries of its direct friends only.

Algorithm 4- **InitializeView**(user u's FOAF)
Input: User u's FOAF file
Output: $local\text{-}view_u$

```
1   For each friend v ∈friends(u) do
2       User u Send its FOAF file to friend v
3       User u Receive friend v's FOAF file
4       User u Extract its friend v's topics of expertise T_v if any
5       If |local-view_u|<view-size then
6           User u Add friend v to its local-view
7       Else
8           user x = selectUser(local-view_u)
9           user u replaces x by v at its local-view
10      End If
11  End For
```

Notice that user u's *local-view* may have entries to non expert users. We keep entries for non expert users, because may be there is a non expert user which has expert friends, and those friends do not have friendship to any other user in the system. Figure 3 shows a snapshot of the system graph G which has 6 users that are expert in two topics t_1 and t_2. The links represents the friendship between users e.g., the links between u_1 and u_2 indicates that u_1 and u_2 are friends. Figure 3 shows also that not all the users are expert e.g., u_3 to u_6 are expert either in topic t_1 or topic t_2, but u_1 and u_2 are non expert. Suppose that u_3 and u_4 do not have friendship to any user in the system except u_2. Thus, if there is no entry refereeing to u_2 at u_1's *local-view*, u_3 and u_4 cannot be reached or known by any user in the system.

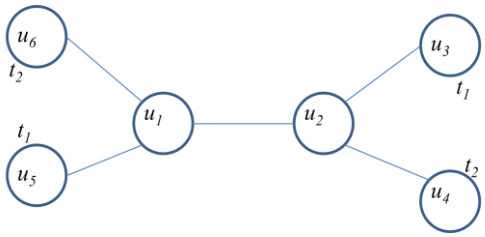

Fig. 3. A snapshot of the system graph *G*

5 Random Gossiping

In this section, we first describe how the P2Prec overlay is constructed and maintained via gossip algorithms. Then, we explain the well known random gossip algorithm [10, 15], and discuss its limitations for P2Prec.

5.1 P2Prec Overlay

P2Prec has an unstructured overly built based on users FOAF files. By gossiping over P2Prec, users may add new friends into their FOAF files, and have a partial view of indirect friends' topics of expertise.

Users use gossip-style communication to construct the P2Prec overlay and exchange a subset of their *local-views* in an epidemic manner [12]. Users also gossip to detect failed users. We choose gossip-style communication for the following reasons. First, the continuous exchange of subset of *local-views* between users enables the building of an unstructured overlay network in a continuous manner, that reflects the natural dynamism of P2P networks and helps provide very good connectivity in the presence of failures or peer disconnections [10]. Second, it provides a reliable way to disseminate information in large-scale dynamic networks, so that users discover new users [16]. Third, a gossip-style communication ensures load balancing during the disseminating of information between users, since all users have the same number of gossip targets and the same exchange period, and thus send exactly the same number of messages [10]. Finally, gossip is scalable, reliable, efficient and easy to deploy [14].

5.2 Random Gossip Algorithm

The basic random gossip algorithm (which we call Rand for short) proceeds as follows: A user *u* (either expert or non expert) acquires its initial *local-view* during the join process. Initially, the *local-view* contains entries of its direct friends only. Then, periodically (with a gossip period noted T_{gossip}), every user *u* exchanges a subset of its *local-view* with one of its direct or indirect friends.

Whenever a user *u* initiates an information exchange, it selects a random contact *v* from its *local-view* to gossip with. Then user *u* selects a random subset of size L_{gossip} -

1, noted *viewSubset*, from its *local-view*, and includes itself into *viewSubset*. Afterwards, *u* sends *viewSubset* to *v*. Similarly, user *u* receives a *viewSubset** of *v*'s *local-view*.

Finally, once a user *u* receives a gossip message, it updates its *local-view* based on the gossip message received. The update process proceeds as follows: 1) the content of the gossip message is merged with the content of the current *local-view* of user *u* and set in a buffer. 2) Using the buffer, *u* selects *view-size* entries randomly and updates its *local-view*. Whenever, user *u* searches for a document, it uses its *local-view* to identify the users in her view that are expert in the topics related to the query.

Rand does not take into account user *u*'s topic of interests during the gossip exchanges. This reduces the possibility of having users in *u*'s *local-view* which are expert in *u*'s topic of interests, and thus reduces the possibility of serving its queries, and reduces quality of query responses. For instance, suppose that user *u* selects user *v* to gossip with, and suppose that *v* has many expert in its *local-view* that are expert in its topic of interests T_v, and suppose that $T_u \cap T_v = 0$. After exchange, user *u* may have many users in its *local-view* that are expert in topics not related to *u*'s topics of interests. In the other hand, suppose that user *u* selects a user *v* to gossip with, and suppose that *v* has many users in its *local-view* that are expert in topics not related to *v*'s topic of interests T_v, and $T_u \cap T_v \neq 0$. The same, after exchange, user *u* may have many users in its *local-view* that are expert in topics not related to *u*'s topics of interests. As a result, user *u* is interested to add to its *local-view* the users which their topics of expertise are related to its topics of interest, and have in their *local-views* many experts that are experts in topics related to *u*'s topic of interests.

Recall that user *u*'s *local-view* may include entries for non expert users, and *u* exchanges a subset of its *local-view* with other user *v* selected randomly from its *local-view*. Thus, exchange messages may contain entries for the non expert users. As a result, user *u*'s *local-view* may be dominated by entries referring to non expert users especially when the number of non expert users in the system is greater than the number of expert users.

6 Semantic Gossiping

In this section, as a first answer to Rand's limitations, we present a new semantic gossip algorithm (called Semt). The goal is to selectively maximize the number of experts at each user's *local-view* where these experts are expert in topics related to the user's topic of interests. First, we explain our criteria for keeping interesting entries in the *local-views*. Then, we present in details the active and passive behavior of Semt.

Recall that our objective is to improve the quality and efficiency of query responses in P2P content sharing systems. Our goal is to let each user *u* maintain a *local-view* in which the topics of expertise of the users in *u*'s *local-view* have high overlap with the topics of *u*'s queries i.e., $\max(T_v \cap T_q)$, where *v* is a user at *u's local-view* and *q* is a query that has been issued by user *u*. Thus, when user *u* initiates a query *q* (see Algorithm 6, Sec. 8), user *u* searches for an expert user $v \in$ *u's local-view* s.t. $T_v \cap T_q \neq 0$. If user *u* finds such expert, *u*'s *hit-ratio* is increased. *Hit-ratio* is defined as the percentage of the number of queries that have been answered.

Moreover, user *u* likes to find many expert users in its *local-view* that can serve its queries, and thus reduces queries response time.

In order to measure user *u*'s *hit-ratio*, we propose the use of a *query-history* that keeps the journal of past queries. With Semt, when a user *u* chooses a contact, it selects a user *v* that has high *hit-ratio*, and close to user *u* in terms of topic of interests. Likewise, *u* includes into *viewSubset* the users that have common topics of interests with *v*, and have high *hit-ratios*. Note that *hit-ratio* can be easily added as an attribute of a *local-view* entry, and part of the gossip message. In the rest of this section, we present our techniques to compute *hit-ratio*, and use it to measure the similarity between users.

6.1 Computing Hit-Ratio

To compute users' *hit-ratios*, we assume that each user *u* maintains a log of limited size, called *query-history*, denoted by H_u. The cardinality of *u*'s *query-history* is denoted by $|H_u|$. *u*'s *query-history* H_u contains a set of entries, each entry referring to a past query *q* that *u* has initiated. Each past query *q* entry in H_u contains:

1. Query topics T_q
2. Query state s_q

Query state s_q takes a value either 1 or -1. When s_q = 1 denotes to *query-success* i.e., user *u* initiates the query *q* and finds at least one expert user in its *local-view* which is expert in query *q* topics T_q. While s_q = -1 denotes to *query-fail* i.e., user *u* has not find any expert user in its *local-view* which is expert in query *q* topics T_q. We use FIFO to replace the past queries once user *u*'s *query-history* has reached its full size $|H_u|$.

Periodically, each user *u* computes its *hit-ratio*. User *u*'s *hit-ratio* represents the percentage of the number of *query-success* in its *query-history* H_u which is:

$$hit\text{-}ratio_u = \frac{\sum_{i=1}^{n} s_{q_i} \in H_u}{|H_u|} \qquad if\ s_{q_i} = 1$$

Where *n* is the total number of past queries available at *u*'s *query-history* H_u.

6.2 Semantic Similarity Function

Recall that each user has a set of topic of interests. Then, we measure the common interest of topics between user *u* and *v*, denoted by *distant(u,v)*, by counting the overlap of their topic of interests. We use the Dice coefficient [54] which is:

$$distant(u,v) = \frac{2|T_u \cap T_v|}{|T_u| + |T_v|}$$

Thus, the *distant(u,v)* represents twice the size of overlap divided by the size of the union of user *u* and *v* topics of interests. We could also use other similarity functions such as cosine, jaccard, etc.

6.3 Semantic Gossip Behaviors

Semt's behavior of each user *u* is illustrated in Algorithm 5. The active behavior describes how a user *u* initiates a periodic gossip exchange message, while the passive

behavior shows how the user *u* reacts to a gossip exchange initiated by some other user *v*. Recall that each user *u* acquires its initial *local-view* during the join process. The initial *local-view* contains at first the entries of its direct friends only.

The active behavior is executed every time unit T_{gossip}. A user *u* initiates a communication message, it computes the similarity distance between itself and each user *v* in its *local-view* (line 4). After that user *u* computes the rank of each user *v* in its *local-view*, denoted by *rank(v)*, which is:

$$rank(v) = hit\text{-}ratio_v + distant(u,v)$$

and adds *rank(v)* to a *RankList* (lines 5 and 6). Notice that *RankList* contains users' entries along with their ranks. Once user *u* has computed the users' ranks and add them in the *RankList*, it selects from the *RankList* a user *v* which has the highest rank to gossip with by using the **selectTop**() method (line 8).

Once user *u* has selected a user *v* to gossip with, it selects L_{gossip} entries from the *RankList* which have the highest rank using **SelectTopEntries** () (line 9). These entries compose user *u viewSubset*. After that user *u* sends to *v* the *viewSubset* along with its topic of interests T_u (line 10). Notice that the entries in *viewSubset* belong to expert users only, because our goal is to exchange users' topics of expertise.

In turn, user *u* will receive a *viewSubset** of user *v*'s *local-view* (line11). Upon receiving *viewSubset**, user *u* computes the rank for each user *v* in *viewSubset** and adds it to the *RankList* (lines 12-16). Recall that *RankList* includes also the rank of the users at *u* current *local-view*. Then, the method **SelectTopEntries**() selects *view-size* entries from the *RankList* which have the highest rank to be as the new *local-view* (line 17).

In the passive behavior, the user *u* waits for a gossip message from a user *v*. Upon receiving a message (line 3), it computes the rank of users in its *local-view* as described (lines 4-8). Then it uses **SelectTopEntries**() to select *viewSubset** of L_{gossip} entries from the *RankList* which have the highest rank (line 9). Then it sends back *viewSubset** to user *v*. After that, it computes the rank of the users in the received *viewSubset* (lines 11-15). Finally, it updates its *local-view* by selecting *view-size* entries from the *RankList* which have the highest rank.

By letting each user *u* select the top ranked entry *v* from its *local-view* as the next gossip contact, may deteriorate the randomness of its *local-view* entries, because it may occur that *v* may remain the same contact for long period of time.

To increase the randomness and prevent user *u* from selecting the same contact *v* for long period of time, we propose that each user *u* stores in a list *L*, the last ℓ recent users that have been selected for gossiping. Recall that user *u* selects from its *local-view* the top ranked user *v* to gossip with. Thus, if user *u* selects a user *v* to gossip with, then user *u* detects that *v* has been selected for gossiping recently i.e., $v \in L$ then user *u* selects the next top ranked user to gossip with and so on.

Moreover, due to the fact that *viewSubset* and gossip contact are not chosen randomly, may reduce the benefits brought by gossiping such as: reduces user's ability to discover new (user or community) of interest. As a consequence, *u* may miss other high interesting contacts. In order to overcome this limitation we propose a semantic two-layered gossiping.

Algorithm 5- **Gossiping**(*local-view*$_u$)
//Active behavior
Input: *local-view*$_u$
Output: updated *local-view*$_u$
1 **Forever do**
2 **wait**(T_{gossip})
3 **For** each user $v \in$ *local-view*$_u$ **do**
4 user *u* **computes** *distant(u,v)*
5 *rank(v) = hit-ratio*$_v$ *+ distant(u,v)*
6 user *u* **adds** <*rank(v)* ,v> to *RankList*
7 **End For**
8 user *v* = **selectTop**(*RankList*)
9 *viewSubset* = **SelectTopEntries**(*RankList*,L_{goosip})
10 User *u* **send** <*viewSubset* ,T_u> to *user v*
11 User *u* **receive** *viewSubset ** from *user v*
12 **For** each user $v \in$ *viewSubset ** **do**
13 user *u* **computes** *distant(u,v)*
14 *rank(v) = hit-ratio*$_v$ *+ distant(u,v)*
15 user *u* **adds** <*rank(v)* ,v> to *RankList*
16 **End For**
17 *Local-view*$_u$ =**SelectTopEntries**(*RankList, view-size*)
//Passive behavior
Input: *viewSubset* of a user *v*; T_v ; *local-view*$_u$
Output: updated *local-view*$_u$
1 **Forever do**
2 **waitGossipMessage**( )
3 **receive** <*viewSubset* ,T_v> from user *v*
4 **For** each user $v \in$*u's local-view* **do**
5 user *u* **computes** *distant(u,v)*
6 *rank(v) = hit-ratio*$_v$ *+ distant(u,v)*
7 user *u* **adds** <*rank(v)* ,v> to *RankList*
8 **End For**
9 *viewSubset** = **SelectTopEntries**(*RankList*,L_{goosip})
10 **send** *viewSubset** to *user v*
11 **For** each user $v \in$ *viewSubset* **do**
12 user *u* **computes** *distant(u,v)*
13 *rank(v) = hit-ratio*$_v$ *+ distant(u,v)*
14 user *u* **adds** <*rank(v)* ,v> to *RankList*
15 **End For**
16 *Local-view*$_u$ =**SelectTopEntries**(*RankList, view-size*)

7 Semantic Two-Layered Gossiping

In this section, we propose a semantic two-layered gossiping (called 2LG) to combine the benefits of Rand (e.g., connected overlay, ability to find new users, etc.) and semantic exchange.

Rand preserves gossiping properties (see Section 5.1), and gives users the ability to discover new users. These new users are then taken into account in Semt to find new interest users.

To enable 2LG we propose the following model. Each user *u* maintains a view for each algorithm. 1) A view for Rand, called *random-view* (first layer), with limited size R_{size}. 2) Similarly, a view for Semt, called *semantic-view* (second layer), with limited size S_{size} s.t. $R_{size}>S_{size}$. Notice that user *u* uses both Rand and Semt views to support its queries.

With 2LG, each user *u* acquires its initial *random-view* during the join process (see Sec. 4.2). Then, user *u* initializes its *semantic-view* by computing the ranks of the users in its initial *random-view* and selects S_{size} entries which have the highest ranks. Afterwards, user *u* periodically (with a gossip period T_{random} and $T_{semantic}$), performs Rand and Semt asynchronously. Notice that, $T_{semantic} >> T_{random}$, because user semantics (topic of interests) are not change rapidly. But we assume that T_{random} is small to capture the dynamicity of the network due to the fact that users in P2P networks are joining and leaving the system continuously.

In 2LG, we adopt Semt (see Algorithm 5) with a modification to take advantage of the *random-view*. Recall that Semt has active and passive behaviors. The modifications are added to the active behavior only. Figure 4 shows the interface that we use for the active behavior of Semt with 2LG, where we keep the lines 1-16 of the active behavior of Algorithm 5, and exploit the entries in the *random-view*. Moreover, Figure 4 shows that the *semantic-view* of a user *u* is updated based on its current Rand and Semt views.

Recall that in the active behavior of Semt a user *u* processes ranking. When a user *u* initiates a gossip exchange, it ranks the users in its *semantic-view*, and adds them in a *RankList* (lines 3-8 at the active behavior of Algorithm 5). As well as, when *u* receives the *viewSubset* it ranks the users in the *viewSubset*, and adds them to the *RankList* (lines 12-18 at the active behavior of Algorithm 5). Thus, the *RankList* includes the rank of the users at *u*'s current *semantic-view* and the ranks of the users in the *viewSubset* that the user *u* has received during the exchange.

In the active behavior of 2LG, each user *u* proceeds as in Semt. However, it also takes into account the user *u*'s *random-view* in the ranking process as follows: User *u* ranks the users in its *random-view*, and adds them to the *RankList* (lines 1-5 at interface 3, Figure 4). Afterwards, *u* selects the S_{size} entries from *RankList* which have the highest rank to be its new *semantic-view* (line 6 at interface 3, Fig. 4).

Interface 3- Active behavior of semantic gossip in two-layered
Input: $semantic\text{-}view_u$ and $random\text{-}view_u$
Output: updated $semantic\text{-}view_u$

Lines 1-16 of the active behavior of Algorithm 5
1 **For** each user $v \in random\text{-}view_u$ **do**
2 user u **computes** $distant(u,v)$
3 $rank(v) = hit\text{-}ratio_v + distant(u,v)$
4 user u **adds** $<rank(v), v>$ to $RankList$
5 **End For**
6 $semantic\text{-}view_u$ =**SelectTopEntries** $(RankList, S_{size})$

Fig. 4. The interface of the active behavior of 2LG

In order to let *isolated-users* benefit from the system, we register each expert user which has joined the P2Prec at a bootstrap server. Once an *isolated-user u* has joined in the system, it periodically contacts an expert user *v* randomly selected from the bootstrap server. If $T_u \cap T_v \neq 0$, *u* asks *v* to send a *viewSubset* of its *semantic-view*, otherwise, it asks *v* to send a *viewSubset* of its *random-view*.

8 Query Routing and Result Ranking

In this section, we first describe the query processing algorithm that we use to generate recommendations. Then, we describe the ranking model we use to order the returned results. Finally, we show how users can manage failure queries.

8.1 Query Processing

We assume keyword queries of the form $q = \{word_1, word_2, ..., word_l\}$, where l is the number of keywords in the query and $word_i$ is the i^{th} keyword in q. Query q can be of type *push* or *pull*. In the push type, the system automatically extracts the keywords of the query q from the documents that are belonging to the user's topics of expertise, such as the most frequent words in the document. In the pull type, user u issues a query q with keywords. For both types, the system extracts q's topic vector, denoted by $V_q = [w_q^{t_1} ....w_q^{td}]$, using LDA as we did for a document. Then query topic(s) $T_q \subset T$ are extracted using **ComputeQueryTopics ()** method as described in Algorithm 1. The query q is considered belonging to a topic $t \in T_q$ if its weight w_q^t in that topic exceeds a certain threshold (which is system defined).

Based on this assumption, each query q issued by a user u has the form $q(word_i, TTL, V_q, T_q, u)$. Algorithm 6 illustrates the behavior of query processing of each user u. In active behavior user u issues a query q: selects from its *local-view* the users which are expert in q's topics T_q. Then user u floods q to them after reducing the query TTL by one (corresponds to line 1). In other words, user u selects each user $v \in u$*'s local-view* s.t. $T_q \cap T_v \neq 0$ and floods q to them. Notice that, *u's local-view* consists from u's *random-view* and u's *semantic-view* in case 2LG is used.

In passive behavior, when user u receives a query q it processes q as follows: First, u selects from its *local-view* the users which are expert in q's topics T_q and floods the query to them while the query TTL does not reach zero (corresponds to

lines 9 and 10). Second, user *u* measures the similarity between query *q* and each document user *u* has (corresponds to lines 3 and 4). The similarity between a document *doc* and a query *q*, denoted by *sim(doc,q)*, is measured by using the cosine similarity [27] between the document topic vector $V_{doc}=[w_{doc}^{t_1}....w_{doc}^{t_d}]$ and the query topic vector $V_q =[\ [w_q^{t_1}....w_q^{t_d}]$ which is:

$$sim(doc,q) = \frac{\sum_{i=1}^{d} w_q^{t_i} * w_{doc}^{t_i}}{\sqrt[2]{\sum_{i=1}^{d} w_q^{t_i} * w_q^{t_i} * \sum_{i=1}^{d} w_{doc}^{t_i} * w_{doc}^{t_i}}}$$

Finally, user *u* returns to the initiator the documents whose similarity exceeds a given (system-defined) threshold (corresponds to lines 5 and 6). Recall that user *u* has assigned access rules to its documents. Thus user *u* returns the documents that can be accessed by the initiator.

Algorithm 6- Query Processing
//Active behavior: **Rout-Query**(*q, local-view*$_u$)
Input: query *q* (*word*$_i$, TTL, V_q, T_q,*u*); *local-view*$_u$
Output: submit *q* to potential experts

```
1   User u Send q to each v ∈ local-view_u s.t. T_q∩T_v ≠0
2   If query-fail  then
3      For each user v∈ local-view_u  do
4         user u retrieve user v query-history H_v
5         If  T_q∩T_qj ≠ 0 and s_qi=1 s.t. q_j∈H_v then
6            User u Send q  to user v
7         End If
8      End For
9   End If
10  If user u Receives docs then
11     User u Ranks docs
12  End If
```

//Passive behavior: **Process-query**(*q, D*$_u$, *local-view*$_u$)
Input: query *q* (*word*$_i$, TTL, V_q, T_q,*u*); *local-view*$_u$
Output: answer set of docs that are related to query *q; u* send *q* to potential experts

```
1   Forever do
2   Receive query q
3   For each doc∈D_u  and doc can be accessed by q's initiator do
4      Sim(q,doc) = CosineSimilarity(V_q,V_doc)
5      If Sim(q,doc) greater than threshold then
6         Send doc  to q's initiator
7      End If
8   End For
9   If q.TTL not equal to zero then
10     u Send q to each v ∈ local-view_u  s.t. T_q∩T_v ≠0
11  End If
```

With such query routing, we avoid sending q to all users' neighbors, thus minimizing the number of messages and network traffic for q. Furthermore, the returned documents are from experts friends (of friends), which increases the confidence of the user in those documents.

8.2 Ranking Returned Results

When an initiator receives the recommended documents from the responders, it merges all documents that come as a response for q. Then it orders them based on their popularity and semantics (corresponds to line 11 in the active behavior of algorithm 6). That is, the rank of a document *doc*, denoted by *rank(doc)*, consists of its semantic relevance with the query q and its popularity:

$$rank(doc) = a * sim(doc, q) + b * pop(doc)$$

Where a and b are scale parameters such that $a + b = 1$ and *pop(doc)* is the popularity of the document *doc* which is equal to the number of responders that have returned document *doc*. The user can specify whether it prefers the highly popular documents or the highly semantically relevant by playing with parameters a and b. Upon receiving recommendation documents, a user u can download a copy of a document, gives a rate to it and includes it in its document D_u.

8.3 Dealing with Queries Failures

We use users' *query-histories* to support failed queries to increase the *hit-ratio*. Whenever, a user u submits a query q, it adds the q topics T_q to its *query-history* along with a state, which indicates if q is successfully submitted or not. If q is successfully submitted, means that u has expert(s) in its *local-view* that are expert in T_q, and thus can serve other queries that their topics are related to T_q.

When a user u submits a query q, query q is considered as *query-fail* if user u does not find any expert user in its *local-view* which is expert in q topics T_q i.e., $T_v \cap T_q = 0$, for each $v \in local\text{-}view_u$. To handle this situation, we exploit the *query-histories* of the users at u's *local-view*.

Recall that each user u maintains a *query-history* H_u. When a user u meets a *query-fail*, u retrieves the *query-history* H_v of each user v in its *local-view*. Then, for each H_v, u computes the intersection between q topics T_q and the topics T_{qi} of each query $q_i \in H_v$ (corresponds to lines 3 and 4 in the active behavior of algorithm 6). If there is a query q_i s.t. $T_q \cap T_{qj} \neq 0$ and $s_{qi}=1$, u sends q to v (corresponds to lines 5 and 6 in the active behavior of algorithm 6). Notice that, we do not use *query-histories* in passive behavior.

9 Experimental Evaluation

In this section, we provide an experimental evaluation of P2Prec to assess the quality of recommendations, search efficiency (cost, and *hit-ratio*), bandwidth consumption, and clustering coefficient. We have conducted a set of experiments using TREC09

[25] and the Wiki vote social network [33]. We first describe the experimentation setup. Then, we evaluate each gossip algorithm and its effect on the respective metrics, and the effect of TTL and *query-histories* on query processing.

9.1 Experimentation Setup

We use the classical metric of *recall* that is used in IR and RSs to assess the quality of the returned results [28]. Recall represents the system ability to return all relevant documents to a query from the dataset. Thus, in order to measure recall, the relevant documents set for each query that have been issued in the system should be known in advance i.e., we need to have relevance judgments for each query that has been issued in the system. Data published by TREC have many relevance judgments. We use the Ohsumed documents corpus [13] that has been widely used in IR. It is a set of 348566 references from MEDLINE, the on-line medical information database, consisting of titles and/or abstracts from 270 medical journals over a five year period (1987-1991). It was used for the TREC09 Filtering Track [25]. It includes a set Q of 4904 queries. The relevant documents for each query q denoted by R_q were determined by TREC09 query assessors. In the experiment, user u issues a query $q \in Q$ and uses P2Prec to possibly retrieve the documents that have been in R_q. The set of documents returned by P2Prec for a user u of a query q is denoted by P_q. Once a user u has received P_q from P2Prec, it can count the number of common documents in both sets P_q and R_q to compute recall. Thus, recall is defined as the percentage of q's relevant documents $doc \in R_q$ occurring in P_q with respect to the overall number of q's relevant documents $|R_q|$:

$$Recall = 100 . \frac{|P_q \cap Rq|}{|Rq|}$$

We use the following metrics to evaluate P2Prec.

- **Communication cost**: the number of messages in the P2P system for a query;
- **Hit-ratio**: the percentage of the number of queries that have been successfully answered.
- **Background traffic:** the average traffic in bps experienced by a user due to gossip exchanges.
- **Clustering coefficient:** the density of connections between peer neighbors. Given a user u, the clustering coefficient of u is the fraction of edges between neighbors (users at u's *local-view*) of u that actually exist compared to the total number of possible edges which is:

$$C.coef = \frac{\sum_{i=1}^{view-size} loacl - view_u \cap loacl - view_{u_i}}{(view - size)(view - size + 1)},\ u_i \in loacl - view_u$$

We extracted the titles and the abstracts of TREC09 documents and removed from them all the stop words (e.g., the, and, she, he, …) and punctuations. Then, we fed them to the GibbsLDA++ software [22], a C++ implementation of LDA using Gibbs sampling, to estimate the document topic vectors V_{doc}. With $|T|$=100 as the number of topics, we ran GibbsLDA++ 2000 times to estimate the document topic vectors V_{doc}. To estimate the query topic vectors V_q, we removed the stop words and punctuations from queries keywords, fed the query keywords left to the GibbsLDA++, and

computed the topics T_q of each query $q \in Q$. We consider that each query $q \in Q$ has one topic $t \in T$ for ease of explanation. We consider that a query q topic t_q is the maximum component of its V_q i.e., the maximum w_q^t.

TREC09 does not have users or authors associated with its documents. Thus, we need a dataset of users which emulates a social network between users. For this purpose, we use the Wiki vote social network [33] and give each user a set of documents from TREC09. In Wiki vote, two users are considered friends if one vote for the other. It consists of 7115 users connected together by 103689 links with an average of 14.57 links per user.

Suppose that the documents popularity follows the zipf law [6]. Then, the document that appears more in Q is given to more users. Similarly, the user that has more friends is given more documents. After distributing the TREC09 documents over the Wiki vote users, these users have 2675240 documents, with an average of 108 documents per user.

We generate a random rate between 0 and 5 for each document a user has and compute the users' topics of expertise from the documents they have rated. We consider that a user may become an expert if it has her number of documents exceeds 108. Also, we assume that a user is expert in topic $t \in T$ if 40% of its documents have high quality in topic t (see Section 4.2). As a result, 43% of users become experts (3060 users) and their expertise range between 1 and 5. We also keep for each user the documents related to her topics of expertise.

P2Prec is built on top of a P2P content sharing system which we generated as an underlying network of 7115 nodes which is equal to the number of users in the Wiki vote network. We use PeerSim [21] for simulation. Each experiment is run for 24 hours, which are mapped to simulation time units.

In order to evaluate the quality of recommendations, we let each user u issue a query after receiving the results from all the users that have received the previous query or after the query has reached a system-specified timeout. The query topic is selected, using zipf law, among u topics of interests T_u. Then we obtain the recommendations for each query and compute recall, communication cost, and response time. In order to obtain global metrics, we average the respective metric values for all evaluated queries.

Table 1. Simulation Parameters

Parameter	Values
Topics	100
TTL	1, 2, 3
Local-view size (*view-size*)	70
Gossip length (L_{gossip})	20
Gossip period (T_{gossip})	30 min
Random-view size (R_{size})	40
Semantic-view size (S_{size})	30
Gossip period for random at 2LG (T_{random})	10 min
Gossip period for semantic at 2LG ($T_{semantic}$)	30 min

Table 1 summarizes the simulation parameters that we have used in the experiments. We do not study the effect of gossip parameters (*view-size*, L_{gossip} and T_{gossip}) on the recommendation quality (see [8] for such study).

9.2 Experiments

We performed our experiments under churn i.e., the network size is changed during the run due to the joining and leaving of users. The experiments start with a stable overlay with 355 users. Then, as experiments are run, new users are joining and some of the existing users are leaving.

We investigate the effect of Rand, Semt and 2LG on the quality of recommendations over the respective metrics. In each experiment, we run one gossip algorithm (Rand, Semt or 2LG) and we use 1 for the TTL of the query. Then, we collect the results for each algorithm after 24 simulation hours. We set TTL to 1 to measure the quality and effectiveness of users' views.

Table 2 shows the results obtained from the experiments. In [8], we showed that the background traffic is affected by gossip period (T_{gossip}) and gossip length (L_{gossip}). We observed that, increasing either T_{gossip} or L_{gossip} increases background traffic while decreasing either T_{gossip} or L_{gossip} decreases it. Thus, Rand and Semt are used with the same gossip parameters (T_{gossip} = 30 minutes, and L_{gossip} = 20), so they consume almost the same bandwidth (2 bps). 2LG consumes more bandwidth, because, four exchange messages are applied in 30 minutes (three exchanges for Rand and one exchange for Semt). Thus, the background traffic in 2LG is four times that of Rand and Semt (8.13 bps).

Rand produces an overlay with a low clustering coefficient. There is a low overlap between a user *u*'s *local-view* and the *local-views* of its neighbors (the users at *u*'s *local-view*). Semt produces a high clustering coefficient. There is a high overlap between users' *local-views*. This is due to the fact that, if a user u_1 is similar to user u_2, and user u_2 is similar to u_2, then most probably u_1 and u_2 are similar, and thus they produce a clique. In 2LG, the clustering coefficient is moderate between those of Rand and Semt gossiping since Semt increases the clique likelihood but Rand increases randomness. Therefore, the clustering coefficient is higher than in Rand but lower than in Semt.

Table 2. Results

Metric	Random	Semt	2LG
Recall	21.2	50.792	33.958
Communication cost	8.17	19.2	15.8
Max. Hit-ratio	0.32	0.804	0.885
Background traffic (bps)	2.04	2.1	8.131
Clustering coefficient	0.064	0.34	0.12

In Figure 5, we show the variation of the recall, communication cost, and *hit-ratio* versus time for the three algorithms. Figure 5(a) shows that the recall at the beginning keeps increasing, and then stabilizes after 12 hours. At the beginning, the network size is small and many expert users are not alive. Thus, many irrelevant documents

are returned, which reduces recall. Semt increases recall by a factor of 2.5 in comparison with Rand and by a factor of 1.67 in comparison with 2LG. That is, because in Semt, a user *u* has in its *local-view* a high number of experts which are expert in topics related to *u*'s queries topics. Thus, when *u* submits a query *q*, the query *q* reaches more experts, and thus more relevant documents are returned.

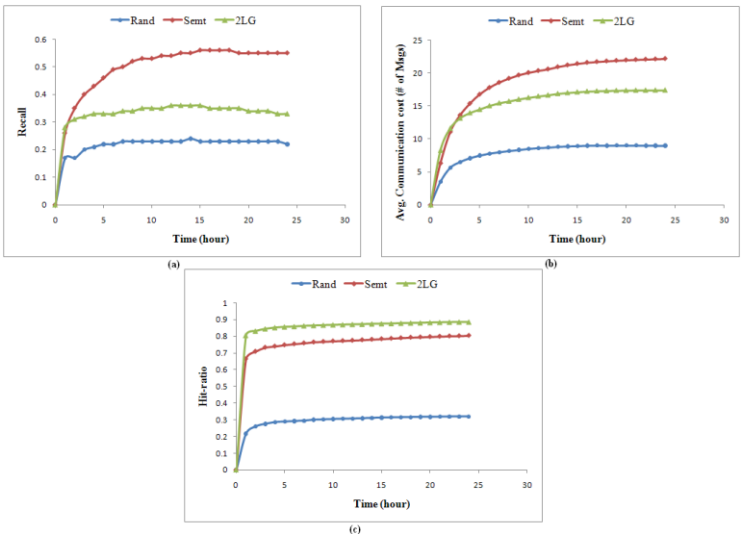

Fig. 5. The variation of recall, communication cost, and hit-ratio versus time

Figure 5(b) shows the communication cost of queries for the three algorithms. We set TTL to 1, so that communication cost represents the number of expert users that serve the query. We observe that Semt has the highest communication cost, because each user *u* includes in its *local-view* a high number of experts which are expert in topics related to *u*'s demands. In Rand, the communication cost is low because each *u* has few experts which are expert in topics related to *u*'s demands. In 2LG the communication cost is a little less than Semt, because the *semantic-view* size (S_{size} = 30) is less than that in Semt (*view-size* = 70).

Figure 5(c) shows the *hit-ratio* for the three algorithms. The maximum *hit-ratio* that has been obtained by Rand is very low (0.32). Under Rand, each user *u* has a few experts which are experts in topics related to *u*'s queries topics. Thus, when *u* submits a query *q*, there is a high probability that *u* does not find an expert in its *local-view* that can serve *q*. In Semt and 2LG, the *hit-ratio* is high because *u*'s *local-view* includes many experts which are expert in topics related to *u*'s demands. Thus, when *u* submits a query *q*, *u* finds many experts in its *local-view* that can serve its query *q*.

9.2.1 Effect of TTL

We investigate the effect of varying TTL on the quality of recommendations over the respective metrics. In each experiment, we run one gossip algorithm and vary TTL. Then, we collect the results for each algorithm under each TTL after 24 simulation

hours. We do not show all the results due to space limitations but we explain our observations.

The TTL variation has significant impact on recall and communication cost especially when Rand is used. When increasing TTL, more users are visited, thus increasing the communication cost and the number of returned documents, which in turn increases recall. In Rand, the communication cost is multiplied by 35 when TTL increases from 1 to 3, while recall is increased from 22% to 53.56%. In Semt, recall is increased from 55% to 73.84%, when TTL increases from 1 to 3, while communication cost is multiplied by 9. Varying TTL does not have significant impact on Semt, due to the fact that users' *local-views* have high overlap. Thus, when a user *u* submits a query *q* to a user *v*, user *v* does not have many users in its *local-view* that do not receive *q* before, because the overlap between *u*'s *local-view* and *v*'s *local-view* is high. However, the TTL variation has moderate impact on 2LG. Hence, recall is increased from 33% to 64.86% when TTL increases from 1 to 3, while communication cost is multiplied by 17. Recall that, in 2LG each user *u* uses its random and semantic view. Thus, when a user *u* submits its query *q* to a user *v*, *v* may find many users in its semantic and random views that have not received *q* yet.

9.3 Effect of Using Query-histories

In this experiment, we study the effect of using users' *query-histories* to support the failed queries. We run 2LG with TTL=1, and we use users' query-histories to support failed queries (see Section 9.3).

Figure 6 shows the effect of using users' *query-histories* to support failed query on recall, communication cost and *hit-ratio*. Using users' *query-histories* increases the *hit-ratio* to 97.9%. That is, each time a user *u* submits a query *q*, there is a high probability to find an expert user to serve its query either from its view or from its neighbors' *query-histories*. Recall that each user *u* maintains in its *semantic-view* the users that are most similar to itself. The queries' topics that have been requested by user *u* are most probably similar to queries' topics that are requested by the users in its *semantic-view*. Thus, when *u* uses the *query-histories* of the users in its views, it most probably finds a user *v* that can serve its query.

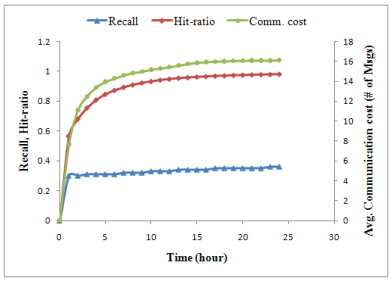

Fig. 6. The effect of *query-histories* on recall, communication cost and *hit-ratio*

Using users' *query-histories* increases recall, because more users are visited and thus more documents are returned. This also increases communication cost, because more users are visited.

10 Related Work

P2Prec is a P2P RS designed to recommend high quality documents by exploiting users' social information. The work most related to ours includes centralized RSs, distributed RSs and social P2P networks.

Centralized RSs. RSs have been used in major applications such as e-commerce, e.g. Amazon.com. There are two kinds of RSs: collaborative filtering (CF) [11] or content-based [3]. CF can recommend products to a user based on the products previously rated by similar users. It works by measuring the similarity between users based on the ratings that have been given by the users over the products [11]. However, CF suffers from data sparsity, i.e. users rate small numbers of products in the system, which leads to several problems. First, a user might have rated little numbers of products and thus the user might not find similar users. Second, a new user which has not rated any products yet will not find similar users to help in finding recommendations [1]. In contrast, P2Prec relies on the explicit friendships between users, which dramatically reduces the possibility that a user will not find friends because it does not have rated enough products yet.

Content-based filtering has been introduced to alleviate CF from the data sparsity problem [3]. Content-based RSs work by suggesting products that are similar to those that the user has seen or rated [3]. The similarity measure is computed between the products the user has seen and the nominated products. Products with high similarity are suggested to the user. However, a user is limited to receive products that are only similar to those it has rated and thus might not explore new interesting topics. In P2Prec, each user maintains a list of friends with different topics of expertise so a user can search a variety of topics even though it is not interested or expert in those topics. All these approaches have relied on the client-server model which has scalability problems.

Distributed RSs. Recently, there has been little work on distributed RSs. The first work is Tveit [31], a P2P RS to suggest recommendations for mobile customers. The system is based on a pure P2P topology like Gnutella and queries that include users' rates are simply propagated by flooding. Miller et al [19] explore P2P RS and propose four architectures including random discovery (similar to Gnutella), transitive traversal, DHT, and secure blackboard. In these architectures, a user aggregates the rating data from the system to make recommendations. However, there are two main drawbacks. First, aggregating users' rates to make recommendations increases the amount of traffic within the network. In contrast, P2Prec lets each user maintain locally a set of users which are expert in a set of topics, so the user uses them to support its query. Second, these systems distribute users' rates independently of any semantic or social structure of the network, while P2Prec is structured based on users' affinity, expertise (documents' semantics) and friendships (users' social).

Social P2P networks. Recent work [2, 32] exploits the social relations between users to self-organize, manage users' information and enhance search in the network. Bai et

al. [2] design a personalize P2P top-k search for collaborative tagging systems. Each user *u* maintains locally a profile which includes the products that it has tagged. In addition, it maintains a *personal network* of users with similar interest along with their profiles. Two users are considered similar if they share a common number of tagged products. In order to find and construct a user *personal network*, two gossip algorithms are used. The first is used as peer sampling to keep the overlay connected. The second is used to gossip users' profiles and measure the similarity between them based on their profiles. Once the user has determined its *personal network*, it stores locally their profiles. When a user issues a query, it uses the profiles of its *personal network* members to process locally its query. Wang et al. [32] propose a P2P RS for television systems on top of Tribler [23]. Each user maintains locally a set of top most similar users (*buddies*) and a set of random peers along with their profiles. Whenever a user selects another user to contact, it first merges the *buddies* with the random peer and ranks them based on the similarity between their profiles with its profile (the similarity between two profiles is measured by counting how many common files they have). Then one user is randomly selected according to a roulette wheel approach. This gives more chance for more similar user to be selected and gives a chance for new user to be explored.

These systems have several problems. First, having users maintain locally the profiles of their *neighbors* (either a user's *personal network* or a user's *buddies*) leads to storage and inconsistency problems. In P2Prec, each user maintains only its documents, thus eliminating this problem. Second, users construct their *neighbors* by gossiping their profiles, which may yield high network bandwidth consumption. In contrast, P2Prec uses gossip only to exchange the topics of expertise between users and the topics of expertise are small. Finally, these systems measure the similarity between users based on the number of common items in their profiles. Unfortunately, this kind of similarity does not capture the context of items. For instance, suppose that a user *u* is interested in topic "*computer science*" and has a set of documents which are related to that topic. Also suppose that another user *v* is interested in "*computer science*" and maintains another set of documents in that topic, but with no document in common. These users will be considered as dissimilar, and thus will not link in the overlay. In P2Prec, the similarity between users is based on their topics of interest.

11 Conclusion

In this paper, we proposed P2Prec, an RS for large-scale data sharing systems that exploits users' social data. P2Prec is useful to recommend to a user high quality documents related to a specific topic from documents that have been seen or created and rated by friends (or friends of friends) who are expert in that topic. To manage users' social data, we use a FOAF file for each user. Each user in the system is automatically assigned topics of expertise, based on a combination of topic extraction from its documents (the documents she shares) and rating. To extract and classify the hidden topics available in the documents, we use the LDA technique. P2Prec is build on top of an unstructured overlay for the sake of scalability and decentralization, and uses gossip algorithms to disseminate user information about experts between friends and friends of friends. P2Prec uses two new semantic-based gossip algorithms (Semt

and 2LG) to let each user aggregate users of common interest and insure randomness in its view.

In our experimental evaluation, using the TREC09 dataset and Wiki vote social network, we showed that using Semt increases recall and *hit-ratio*. This is because each user maintains in its *local-view* a high number of experts which can serve its demands. Using Rand decreases the overlap between users' *local-views* and thus, increases the randomness. Using 2LG exploits the advantages of Rand and Semt. It increases recall and *hit-ratio* by a factor of 1.6 and 2.8, respectively, compared with Rand and reduces the overlap between users' *local-views* by a factor of 2.8 compared with Semt.

Using gossip style communication to exchange the topics of expertise (especially in Semt) increases the system's ability to yield acceptable recall with low overhead in terms of bandwidth consumption. Furthermore, it increases the query *hit-ratio* because gossiping brings more related experts in a user *local-view*, thus reducing the possibility that the user does not find users satisfying its query.

12 References

1. Adomavicius, G., Tuzhilin, A., Toward the next generation of recommender systems: a survey of the state-of-theart and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749, 2005.
2. Bai, X., Bertier, M., Guerraoui, R., Kermarrec, A. M., Toward personalized peer-to-peer top-k processing. In: *International workshop on Social Network Systems*, 2009.
3. Billsus, D. Pazzani, M. J., Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, 46–54, 1998.
4. BitTorrent P2P File Sharing. http://www.bittorrent.com/index.html.
5. Blei, D. M., Ng, A. Y., Jordan, M. I., Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
6. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S., Web Caching and Zipf-like Distributions: Evidence and Implications. *In INFOCOM*, 1999.
7. Callan, J., Distributed Information Retrieval. In W. B. Croft, editor, Advances in Information Retrieval,127-150, Kluwer Academic Publishers, 2000.
8. Draidi, F., Pacitti, E., Valduriez, P., Kemme, B., P2Prec: a Recommendation Service for P2P Content Sharing Systems. *Bases de Données Avancées*, 2010.
9. eMule project. http://www.emule project.net.
10. Gavidia, D., Voulgaris, S., Steen, M., Cyclon: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and System Management*, 13(2), 2005.
11. Goldberg, D., Nichols, D., Oki, B., Terry, D., Using Collaborative Filtering to Weave an Information Tapestry. Communications of the ACM. 35(12):61–70, 1992.
12. Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Demers, A., Greene, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. *ACM Symp. on Principles of Distributed Computing (PODC)*, 1-12, 1987.
13. Hersh, W.R., Buckley, C., Leone, T., Hickam, D.H., Ohsumed: An interactive retrieval evaluation and new large test collection for research. *In Proceedings of the ACM SIGIR*, 192-201,1994.
14. Jelasity, M., Montresor, A., Epidemic-style Proactive Aggregation in Large Overlay Networks. *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 102-109, 2004.
15. Jelasity. M., Voulgaris, S., Guerraoui, R., Kermarrec, A. M., VanSteen, M., Gossip-based peer sampling. *ACM Trans. Comput. Syst*., 25(3):8, 2007.

16. Kermarrec, A.M., Eugster, P.T., Guerraoui, R., Massoulieacute, L., Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, 37(5): 60-67, 2004.
17. Kruk, S. R., Foaf-realm - control your friends' access to the resource. *In FOAF Workshop proceedings*, 2004.
18. LiveJournal social network. http://livejournal.com
19. Miller, B.N., Konstan, J.A., Riedl, J.: PocketLens, Toward a Personal Recommender System. *ACM Trans. on Information Systems*, 22(3):437-476, 2004.
20. Ogasawara, S,. Paulino, C., Gresta, L., Murta, P., Werner, C., Mattoso, M., Experiment Line: Software Reuse in Scientific Workflows. *SSDBM*, 264-272, 2009.
21. Peersim p2p simulator. http://www.peersim.sourceforge.net
22. Phan, X.-H., http://gibbslda.sourceforge.net
23. Pouwelse, J.A., Garbacki, P., Wang, J., Bakker, A.,Yang, J., Iosup, a., Epema, D.H.J., Reinders, M., Vansteen, M.R., Sips, H.J., TRIBLER: a social-based peer-to-peer system. *In IPTPS'06, Santa Barbara*, 2006.
24. Qiao, Y., Bustamante, F.E., Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use. *In Proceedings of the USENIX Annual Technical Conference (ATEC)*, 341–355, 2006.
25. Robertson, S., Hull, D.A., The TREC-9 filtering track final report. In: *Proceedings of the 9th Text REtrieval Conference TREC-9*, 25-40, 2001.
26. Ruthven, I., Lalmas, M., A Survey on the Use of Relevance Feedback for Information Access Systems. *The Knowledge Engineering Review*, 18, 95-145,2003
27. Salton, G., A Theory of Indexing. *Regional Conf. Series in Appl. Math., Soc. For Indust. And Appl. Math*., Philadelphia, Pennsylvania, 1975.
28. Sarwar, B., Karypis, G., Konstan, J., Riedl, J., Analysis of Recommendation Algorithms for e-commerce. ACM Conf. on Electronic Commerce, 158–167, 2000.
29. Sinha, R., Swearingen, K., Comparing Recommendation made by Online Systems and Friends. *In the Proc. Of the DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*, Ireland, 2001
30. The friend of a friend (FOAf) project. http://www.foaf-project.org.
31. Tveit, A., Peer-to-Peer Based Recommendations for Mobile Commerce. *Proceedings of the International Workshop on Mobile Commerce*, 26–29, 2001.
32. Wang, J., Pouwelse, J.A., Fokker, J., Reinders, M.J.T., Personalization of apeer-to-peer television system. *In Proc.of Euro ITV2006*, Athens, 2006.
33. Wikipedia adminship vote network. http://snap.stanford.edu/data/wiki-Vote.html.