Leveraging Social and Content-based Recommendation in P2P Systems

Fady Draidi, Esther Pacitti, Michelle Cart, Hinde Lilia Bouziane
INRIA & LIRMM, Montpellier, France
{fady.draidi, esther.pacitti, cart, hinde.bouziane}@lirmm.fr

***Abstract*-We focus on peer-to-peer (P2P) content recommendation for on-line communities, where social relationships between users can be exploited as a parameter to increase the trust of recommendation. Most of the existing solutions establish friendship relationships based on users behavior or declared trust. In this paper, we propose a novel P2P recommendation approach (called F2Frec) that leverages content and social-based recommendation by maintaining a P2P and friend-to-friend network. This network is used as a basis to provide useful and high quality recommendations. Based on F2Frec, we propose new metrics, such as usefulness and similarity (among users and their respective friend network), necessary to enable friendship establishment and to select recommendations. We define our proposed metrics based on users' topic of interest and relevant topics that are automatically extracted from the contents stored by each user. Our experimental evaluation, using the TREC09 dataset and Wiki vote social network, shows the benefits of our approach compared to anonymous recommendation. In addition, we show that F2Frec increases recall by a factor of 8.8 compared with centralized collaborative filtering.**

Keywords-P2P systems; personalization; recommendation; gossip protocols; social networks.

I. INTRODUCTION

We focus on Peer-to-Peer (P2P) large scale content sharing for on-line communities. For instance, in modern e-science (e.g., bio-informatics, physics and environmental science), scientists must deal with overwhelming amounts of contents (experimental data and documents, images, etc.) produced and stored in his workspace that they are willing to share within a community or with specific friends without relying in a centralized server.

Peer-to-Peer (P2P) networks, offers scalability, dynamicity, autonomy and decentralized control. Locating contents based on contents ids in a P2P overlay network is now well solved (see [4]). However, the 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 is a recommendation service that, given a query, can recommend relevant documents by exploiting user information.

Sinha et al. [11] have shown that users prefer the advices that come from known friends in terms of quality and trust, because users typically *trust* their friends' advices. The emersion of Web2.0 and the growing popularity of online social networks have encouraged exploiting users' social data in P2P systems. In existing P2P solutions, friendship links are extracted from users' behaviors [6] or are established based on explicit trust declaration [8]. To enrich these solutions, we consider that users that store similar contents may be potentially friends with a specific declared trust level with respect to the relevance of a user in a specific topic. Thus, our decentralized recommendation approach leverages content-based and social-based recommendation over a distributed graph, where each node represents a user labelled with the contents it stores and its topics of interests. As a basis for recommendation, we propose new social metrics such as similarities (among users and their respective friend network) and usefulness of a user with respect to a friend or query taking into account the declared trusts. These measures are defined based on user topics of interest and relevant topics that are automatically extracted from the contents they store. Notice that a user is considered *relevant* in a specific topic *t* if it has a sufficient amount of content with high probability related to *t.* Then this user will be relevant to serve queries related to *t.* also a user *v* is considered *useful* to a user *u*, if *v* is relevant in topics that *u* is interested in.

We implement friendship networks using concepts from the Friend-Of-A-Friend (FOAF) project. FOAF provides an open, detailed description of profiles of users and the relationships between them using a machine-readable syntax. We use FOAF files to support users' queries. To establish friendship and disseminate recommendation, we rely on gossip protocols [3] as follows: At each gossip exchange, each user *u* checks its gossip *local-view* to enquire whether there is any relevant user *v* that is useful to *u*, and whether its friendship networks have high overlap with *u*'s friendship network. If it is the case, a demand of friendship is launched among *u* and *v* and the respective FOAF files are updated accordingly.

Whenever a user submits a keyword query, its FOAF file is used as a directory to redirect the query to the top-k most adequate friends taking into account similarities, relevance, usefulness and trust. In our previous work [3], we focused on P2P anonymous recommendation exploiting different types of gossip protocols.

In this paper, we propose F2Frec, a new social-based approach for recommendation that facilitates the construction and maintenance of P2P social network and exploits social metrics to provide recommendations. Our major contributions are:

- We introduce new social metrics to suggest friends and detect if a friend is relevant and useful to provide recommendations.
- We propose an efficient query routing algorithm that takes into account the social metrics to select, in a top-k approach, the most appropriate friends to provide recommendation.

- Once the best recommendations are provided, we propose to rank them by taking into account the semantic similarities, content popularity, distance and trust between query's initiator and responders.
- We provide an experimental evaluation using real data sets that demonstrates the efficiency of F2Frec over the TREC09 [10] and Wiki vote social network [12] compared to anonymous P2P recommendations and centralized recommendation.

The rest of this paper is organized as follows. Section II provides a general overview of F2Frec. Section III presents our social metrics and how we manage friendship establishment. Section IV describes our solution for retrieving recommendations over F2Rrec given a key-word query. Section V gives our experimental validation that compares F2Rrec with centralized collaborative filtering. Section VI discusses related work. Section VII concludes.

II. General Overview of F2F Recommendation

Our recommendation model is expressed based on a graph $G = (D,U,E,T)$, where D is the set of shared documents, U is the set of users $u_1,...u_n$ corresponding to autonomous peers $p_1,...p_n$, 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 domain of topics. Each user $u \in U$ is associated with a set of topics of interest $T_u \subset T$, and a set of relevant topics $T_u^r \subset T_u$ extracted locally from the documents u has rated. The rating that has been given by a user u on document doc is denoted by $rate_{doc}^{u}$.

In our approach, we use Latent Dirichlet Allocation (LDA) [2] to automatically extract the topics in the system, which in turn are used to extract users' relevant topics of interest. In F2Frec, LDA processing is done in two steps: the training at a global level, and inference at the local level. The global level is given to the bootstrap server (BS) that aggregates a sample set of M documents from F2Frec participant peers. Then, BS runs the LDA classifier to get a set $T = \{t_1,..t_k\}$ of topics, where k is the number of topics. Each topic $t \in T$ contains a set of Z words, where Z is the number of unique words in M, and each word $z \in Z$ is associated with a weight value w_z^t between 0 and 1. The w_z^t represents how much the word $z \in Z$ is related to t. At the local level, user u performs LDA locally to extract the topics of its local documents, using the same set of topics T that were previously generated at the global level. LDA provides a vector of size k for each document doc, $V_{doc} = [w_{doc}^{t1},...,w_{doc}^{tk}]$, where w_{doc}^{t} is the weight of each topic $t \in T$ with respect to doc.

Users' relevant topics of interest are extracted based on a combination between documents' semantics and ratings. Since we focus on on-line communities, we safely assume that users are willing to rate the documents they store. Once a user u extracted the V_{doc} for each $doc \in D_u$, it multiplies the $V_{doc} = [w_{doc}^{t1},..,w_{doc}^{tk}]$, by the rating $rate_{doc}^{u}$. Then, user u identifies for each topic $t \in T$ only the documents that are highly related to t. A document doc is considered highly related to topic t, if its weight in that topic w_{doc}^{t} multiplied by its rating $rate_{doc}^{u}$ exceeds a threshold value. Next, u counts how many documents are highly related to each topic $t \in T$. User u is considered interested in topic $t \in T_u$ if a percentage y of its local documents are highly related to topic t. Finally, u is considered a *relevant user* in topic $t \in T_u^r$ if it is interested in t and has a sufficient amount x (system-defined) of documents that are highly related to topic t.

Each user $u \in U$ maintains locally a FOAF file that contains a description of its personal information, and friendship network, denoted by $friends(u)=\{f_1, f_2,...f_n\}$. Personal information includes the extracted topics of interest, where each topic of interest $t \in T_u$ is associated with a Boolean value that indicates whether u is relevant in that topic. Friends' information includes friends' names, links (URI) to their FOAF files, relevant topics of interest, and trust levels. The trust level between user u and a friend v, denoted by $trust(u,v)$, is a real value within [0, 1] and represents the faith of user u in its friend v. The trust level between user u and its friend v can be obtained explicitly [8] or implicitly [7].

Furthermore, each user $u \in U$ establishes new friendships with users that are *useful* to u's demands or have friendship networks with high overlap with u's friendship network. A user v is considered *useful* to a user u, if v is a relevant user and a certain amount of v's relevant topics T_v^r are of interest for u. User u exploits its useful friends (of friends) for recommendations. Notice that, if a friendship acquaintance exists between users u and v, u implicitly recommends its documents to v and vice-versa, in related topics. More precisely, if there is a friendship path between users u and v, $path(u,v)=\{(u, v_i), (v_i,v_j),...,(v_k, v)\}$, then u can recommend its documents related to their topics of interest to v and vice-versa.

Queries are expressed through key-words, and mapped to topic(s) T_q using LDA. Moreover, queries are associated with a TTL (Time To Live), and routed recursively on a distributed top-k algorithm: Once a query is received by any user, it is forwarded to its top-k best friends by taking into account usefulness and trust. A response to a query q is a recommendation provided in a ranked list and defined as:

$$recommendation_q = rank(rec_q^1(doc_1),... rec_q^n(doc_j)) \quad (1)$$

Different recommendations may be given for the replicas of a document doc_i. The $recommendation_q$ is ordered based on a ranking function, that ranks each $rec_q^v(doc_i)$ according to its relevance with q, its popularity, and the distance and trust between the q initiator and responder v. More details on query processing and recommendations ranking are given in Section IV.

The trust value between a query's initiator u and a responder v, denoted by $trust_q(u,v)$, is computed during query processing. The path of a query q between u and v can be represented as $path_q(u,v)=\{(u, v_i), (v_i,v_j), (v_j, v)\}$, and the trust value between u and v can be computed by multiplying the trust values among directs friends along the $path_q(u,v)$, which is:

$$trust_q(u,v) = \prod_{v_i,v_j \in path_q(u,v)} trust(v_i, v_j) \quad (2)$$

III. Friend to Friend Recommendation

The goal is to let each user explicitly establish friendship with useful users, so that it can exploit them for recommen-

dation. First, we present the similarity metrics we propose. Then, we present the data structures and algorithms for friendship establishment.

A. Metrics

We compute the similarity distance between *u* and *v* based on their friendship networks and relevant topics of interest. We measure the similarity distance between *u* and *v* based on their friendship networks, denoted by $distance_{fri}(u,v)$, by counting the overlap of their friends. We use the dice coefficient, which is:

$$distance_{fri}(u,v) = \frac{2|friend(u) \cap friend(v)|}{|friend(u)|+|friend(v)|} \quad (3)$$

We could also use other similarity functions such as cosine, jaccard, etc. We use $distance_{fri}(u,v)$ as a measure for the implicit trust between *u* and *v*.

We measure the common interest of topics between user *u* and *v*, denoted by $distance_{intr}(u,v)$, by counting the overlap of their topic of interests. We use the dice coefficient, which is:

$$distance_{intr}(u,v) = \frac{2|T_u \cap T_v|}{|T_u|+|T_v|} \quad (4)$$

Notice that user *u* and *v* may be similar in terms of topics of interest. However, *v* may not be useful for *u*, because the topics of interest of *u* are not related to *v*'s relevant topics. Therefore, we measure how much *v* is useful to *u*, denoted by $useful(u,v)$, by counting the overlap between *u*'s topics of interest T_u and *v*'s relevant topics T_v^r. Similarly, we use the Dice coefficient to measure $useful(u,v)$:

$$useful(u,v) = \frac{2|T_u \cap T_v^r|}{|T_u|+|T_v^r|} \quad (5)$$

We measure the final similarity distance between *u* and *v*, denoted by $sim(u,v)$, by combining $distance_{fri}(u,v)$ with $useful(u,v)$ in a weighted approach as follows:

$$sim(u,v) = \alpha * useful(u,v) + (1-\alpha) * distance_{fri}(u,v) \quad (6)$$

The parameter α is used to adjust whether *u* prefers to establish friendship with users that are highly useful to its queries, or with users that their friendship networks are highly overlapped with *u*'s friendship network. As α values become close to 1, the usefulness of users play a more important role in the final similarity distance $sim(u,v)$.

Also, we use the Dice coefficient to measure how much a relevant user *v* is useful to a query *q*:

$$useful(q,v) = \frac{2|T_q \cap T_v^r|}{|T_q|+|T_v^r|} \quad (7)$$

If $useful(q,v) \neq 0$, then the relevant user *v* can give recommendations for *q*.

B. Friendship Establishment

Each user *u* exploits its gossip *local-view* to establish friendship. For each gossip cycle, *u* goes through each user entry $v \in local\text{-}view_u$, and evaluates whether *v* may be suggested for friendship as follows: User *u* computes the similarity distance $sim(u,v)$ as described in Section III.A. User *v* is suggested to *u* for friendship under some conditions, taking into account the degree of similarity $sim(u,v)$, the $distance_{fri}(u,v)$, the $distance_{intr}(u,v)$, $useful(u,v)$, and *v*'s relevant topics, etc. If *u* has accepted to establish friendship with *v*, user *u* sends a message to *v*, denoted by msg_{req}, asking *v* for a friendship. Then, *u* adds *v* to a *waitList* list, waiting for friendship confirmation.

Afterwards, user *u* receives a reply message, denoted by msg_{rep}, from each user $v \in waitList$. If user *v* has accepted to establish friendship with *u* i.e., $msg_{rep} = accept$, *u* stores *v*'s information in its FOAF file. The information for the new friend *v* includes *v*'s relevant topics of interest, a trust value $trust(u,v)$ between *u* and *v*, and link to *v*'s FOAF file. Notice that the $trust(u,v)$ is assigned explicitly by *u* [8].

IV. Query Processing Based on FOAF File

In this section, we describe our query processing algorithm to generate recommendations. Next, we describe the ranking model we use to order the returned recommendations.

A query is defined as $q(word_i, TTL, V_q, T_q, trust_q(u,v), k)$, where $word_i$ is a list of keywords, TTL is the time-to-live value, V_q is query *q*'s topic vector. Query *q*'s topic vector, $V_q = [w_q^{t1},..,w_q^{tk}]$, is extracted using LDA. Then, query topic(s) $T_q \subset T$ are computed, where *q* is considered to belong to a topic $t \in T_q$ if its weight w_q^t in that topic exceeds a certain threshold (which is system-defined). The $trust_q(u,v)$ is the trust level between *u* and a responder *v*. The value *k* is the parameter for top-k redirection.

Each time, a user *u* issues a query *q*, it proceeds as follows: First, it computes how much each useful friend $v \in friend(u)$ is useful to *q*. Then, *u* computes the rank of *v*, denoted by $rank(v)$. The rank of a useful friend *v* for *u* depends on the usefulness of *v* for *q*, and the trust level between *u* and *v*. Accordingly the $rank(v)$ is defined as:

$$rank(v) = trust(u,v) * useful(q,v) \quad (8)$$

Once *u* has computed the rank of each useful friend *v*, it adds $rank(v)$ to a *RankList* that contains the useful friends' addresses along with their ranks. Then, it selects the top-k useful friends from the *RankList* with highest rank, and adds them to *topkList*. Then, *u* forwards *q* to each useful friend $v \in topkList$, attaching to *q* the trust value $trust_q(u,v)$, and reducing the query TTL by one. Note that the value of $trust_q(u,v)$ is equal to the value of $trust(u,v)$, because *v* is a direct friend of *u*. Also the useful friend *v* with the highest rank is the useful friend that is most useful to *q*, and has the highest trust level with *u*.

Once user *u* receives the recommendation information from the responders, it ranks those recommendations and presents them in an ordered list (see Section IV.A).

When a user *v* receives a query *q* that has been initiated by a user *u*, it processes *q* as follows: First, it measures the similarity between query *q* and each document *v* has locally. The similarity between a document *doc* and *q*, denoted by $sim(doc,q)$, is measured by using the cosine similarity between the document topic vector $V_{doc} = [w_{doc}^{t1},...,w_{doc}^{tk}]$ and the query topic vector $V_q = [w_q^{t1},...,w_q^{tk}]$, which is:

$$sim(doc,q) = \frac{\sum_{i=1}^{d} w_q^{t_i} * w_{doc}^{t_i}}{\sqrt{\sum_{i=1}^{d} w_q^{t_i} * w_q^{t_i} * \sum_{i=1}^{d} w_{doc}^{t_i} * w_{doc}^{t_i}}} \quad (9)$$

Second, *v* returns to the query's initiator *u* the recommendations for the documents whose similarity exceeds a given (system-defined) threshold.

Finally, *v* selects from its friends the top-k useful friends that have the highest rank, and adds them to the *topkList* if the query's TTL is not yet zero. Then, *v* computes the trust value $trust_q(u,x)$ for each useful friend $x \in topkList$ based on Equation 2. Then *v* attaches $trust_q(u,x)$ to *q*, and forwards *q* to *x* after reducing TTL by one.

With such query routing, we avoid sending *q* to all friends, thus minimizing the number of messages and network traffic for *q*. In addition, we send the query to friends that are most useful and trustful.

A. Ranking Recommendations

Recall that the result of a query *q* submitted by a user *u* is $recommendation_q = rank(rec_q^{v1}(doc_1),\ldots, rec_q^{v}(doc_i))$, where $rec_q^{v}(doc_i)$ is the recommendation that has been given for a document doc_i from a responder *v*. We rank $rec_q^{v}(doc_i)$ based on the semantic similarity between *q* and doc_i, the popularity of doc_i, and the distance and trust between *u* and the responders of doc_i. Accordingly, $rec_q^{v}(doc_i)$ that has been received from responder *v* includes $sim(doc_i,q)$, *v*'s topics of interest T_v and the $trust_q(u,v)$. The rank of a $rec_q^{v}(doc)$, denoted by $rank(rec_q^{v}(doc))$, is defined as:

$$rank(rec_q^{v}(doc)) = a * sim(doc, q) + b * \frac{\sum_{v \in R} distance_{intr}(u, v) * trust_q(u, v)}{|R|} + c * pop(doc) \quad (10)$$

Where *a*, *b* and *c* are scale parameters, *pop(doc)* is the popularity of *doc*, and |*R*| is the number of responders that have recommended *doc* to the initiator *u*. The popularity is equal to the number of replicas of *doc* in F2Frec. The user can specify whether it prefers highly popular documents, documents that are highly semantically relevant to *q*, or documents that come from highly similar users, by adjusting parameters *a*, *b* and c. Upon receiving the recommended documents, user *u* can download a copy of a document, rate and include it in its document set D_u.

V. Experimental Evaluation

In this section, we provide an experimental validation of F2Frec to assess the quality of recommendations, search efficiency (cost, and *hit-ratio*), and the average number of friends. We conducted a set of experiments using TREC09 [10] and the Wiki vote social network [12]. We first describe the experimentation setup. Then, we evaluate the effect of friendship establishment on the performance of F2Frec. Finally, we compare F2Frec with centralized collaborative filter.

A. Experimentation Setup

We use the classical metric of *recall* that is used in information retrieval and recommender systems to assess the quality of the returned recommendations. 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. Data published by TREC have many relevance judgments. We use TREC09 filtering track [10], a set of 348566 references from MEDLINE, the on-line medical information database, consisting of titles and abstracts from 270 medical journals over a five year period (1987-1991). It includes also a set *Q* of 4904 queries. The relevant documents for each query $q \in Q$, denoted by R_q, were determined by TREC09 query assessors.

In the experiments, user *u* issues a query $q \in Q$ and uses F2Frec to possibly retrieve the documents that are in R_q. The set of documents returned by F2Frec for a user *u* and a query *q* is denoted by P_q. Once a user *u* has received P_q from F2Frec, 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 $|R_q|$:

$$recall = 100 . \frac{|P_q \cap R_q|}{|R_q|} \quad (11)$$

In addition we use the following metrics to evaluate F2Frec.

- **Communication cost:** the number of messages in the P2P system for a query.
- **Background traffic:** the average traffic in bps experienced by a user due to gossip exchanges.
- **Hit-ratio**: the percentage of the number of queries that have been successfully answered.
- **Average number of friends in the network:** the total sum of the number of friend of all users divided by the size of the network (total number of users).

We extracted the titles and abstracts of TREC09 documents and removed from them all the stop words (e.g., the, and,..). Then, we fed them to the GibbsLDA++ software [9], a C++ implementation of LDA using Gibbs sampling, to estimate the document topic vectors V_{doc}. With $|T|=100$ as the number of topics. To estimate the query topic vectors V_q, we removed the stop words from queries keywords, fed the query keywords left to GibbsLDA++, and computed the topics T_q of each query $q \in Q$. For ease of presentation, we consider that each query $q \in Q$ has one topic $t_q \in T$.

We use the Wiki vote social network [12] to give randomly each user a set of documents from TREC09. Wiki vote considers that two users are considered friends if one votes for the other. It consists of 7115 users connected together by 103689 links with an average of 14.57 links per user. After distributing the TREC09 documents over the Wiki vote users, we get a total of 6816170 documents, with an average of 958 documents per user.

We generate a random rating between 0 and 5 for each document a user has and compute the users' topics of interest from the documents they have rated. We consider that each user *u* is interested at least in one topic and relevant at least for one topic. Also *u* is interested in at most 10 topics and relevant for 5 topics at most.

F2Frec is built on top of a P2P content sharing system that we generated as an underlying network of 7115 nodes, which is equal to the number of users in the Wiki vote net-

work. We use PeerSim 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 computing the previous query or after a system-specified timeout. Then we obtain the result for each query and compute the respective metric values. In order to obtain global metrics, we average the respective metric values for all evaluated queries. We let each user *u* establish new friends after each time it performs a gossip.

B. *Experiments*

We first investigate the effect of friend establishment on the performance of F2Frec over the respective metrics. Second, we compare F2Frec with a centralized collaborative filter. For the gossip parameters (gossip period C_{gossip}, gossip message L_{gossip}, and *view-size*), we use 30 minutes for C_{gossip} (simulation time units), 10 for L_{gossip}, and 50 for *view-size* (in [3] we showed that this setting provided good quality of recommendations with acceptable network traffic). We use 1 for the TTL of the query, and query is forwarded to each friend *v* that is useful to query, in order to measure the quality and effectiveness of friendship establishment.

Then, we collect the results for each experiment after 24 simulation hours. We set TTL to 1 to measure the quality and effectiveness of friendship establishment. All experiments are performed 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.

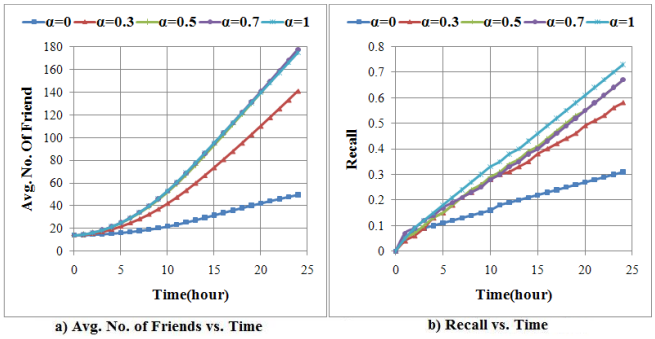
Friendship Establishment. In this experiment, we vary the value of α between 0 and 1, in order to investigate the trade-off of usefulness and friendship distance based on the Equation 6. In addition, we investigate the effect of friendship establishment on the performance of F2Frec over the respective metrics. In each experiment, each user $u \in U$ gets its initial friends from the Wiki social network, then *u* runs the F2Frec algorithm to establish new friends.

Table 1 shows the results obtained after 24 hours of running the F2Frec algorithm. We can see that the average number of friends increases from 49.7 to 174.6 when increasing α from 0 to 1. Combining users' usefulness with friend networks increases the likeness between users. Thus more new friends are added to users' FOAF files. We also observe that recall, communication cost, hit-ratio and background traffic are correlated to the average number of friends. The communication cost increases because more useful friends are visited. Visiting more useful friends increases the relevant documents returned, and thus greater recall is achieved. Also, hit-ratio increases as long as the average number of friends also increases, because there is a higher probability to find a useful friend to serve a query. However, bandwidth consumption increases because increasing the number of friends implies the increase of the size of the gossip entries, increasing the size of the gossip messages. As a result the bandwidth consumed is increased.

In Figure 1, we show the variation of average number of friends and recall versus time under different values of α. We observe that combining the usefulness of users with friendship networks increases the possibility of finding new friends (Fig. 1(a)). When the value of α is equal to 0, the final friendship establishment depends on the overlap between users' friends only. This depends on the density of the links in the network graph. In our benchmark, the overlap between friend networks is low, and thus the average number of friends is low, which causes low recall. However, the recommended documents in this case have more confidence and quality, and users are more satisfied with those recommendations. This is because they are recommended by trusted friends.

TABLE 1. RESULTS OBTAINED BY F2FREC OVER THE RESPECTIVE METRICS

α	Max. recall	Max. Com. cost	Max. Hit-ratio	Max. Avg. background traffic (bps)	Max. Avg. Friend
0	0.31	20	0.61	12.4	49.7
0.3	0.58	38.3	0.94	17.4	141.1
0.5	0.67	46	0.977	19	177.6
0.7	0.67	47	0.98	18.7	177.6
1	0.73	46.5	0.98	18.5	174.6



a) Avg. No. of Friends vs. Time b) Recall vs. Time

Figure 1. F2Frec performance over respective metrics

When the value of α is equal to 1, friendship establishment depends on the usefulness of users only. Each time a user *u* performs gossip, new relevant users are added to its *local-view*. Thus, *u* finds new relevant users that are useful to its demand, and then establishes friendship with them. Therefore, more friends are added at *u*'s FOAF file. As a result, the average number of friends is increased. While the values of α increase between the two extremes, *u* finds new relevant users that are useful to its demand, and establishes friendship with them. Accordingly, its friend list is increased. Then, the possibility of overlap between users' friends increases as well. As a result, the possibility of establishing new friendship increases.

We observe that the recall achieved by α=1 is greater than that with α=0.7 or 0.5, even though the average number of friends are almost identical (Fig. 1(b)). When α=1, friendship establishment depends on users usefulness only. Accordingly, each user *u* establishes new friendship with relevant users that are more useful to its demands.

For the other simulations, we set α=0.5, because this setting leverages users' usefulness and friendship networks, and provides reasonable results with acceptable overhead in terms of background traffic.

Social Effect. We compare F2Frec with a centralized collaborative filter [1] with respect to recall. In order to recommend a set of documents that a user *u* may like, we compute the similarity between a user *u* and all the users in the system. Then, we select a set of users, noted *neighbors(u)*, which are the top-k similar to *u*. We use the cosine similarity to extract a user's *neighbors*, based on the ratings that are given by the users over the documents they have seen or created. Once the similarity between *u* and each user *v* has been computed, we select the top 178 similar users as the *neighbors(u)*. Once users' *neighbors* are extracted, we run the system and generate recommendations for each user from its *neighbors*, and then compute the average recall of all users. The recommendations for each user *u* are generated as follows: First, *u* randomly selects a query $q \in Q$ s.t. $t_q \in T_u$. Then *u* forwards *q* to each member in its neighbors. Each *neighbor* receives *q*, returns to *u* all the documents that their similarity with *q* exceed 0.5. We select the top 178 similar users as the *neighbors(u)*, and $t_q \in T_u$, to be identical with F2Frec.

We observe that the similarity measure is time consuming as it takes about 38 hours to compute the similarity between the 7115 users with 6816170 documents. This time increases exponentially as the numbers of documents and users increase.

Results. We observe that the average recall achieved by collaborative filter is equal to 0.076. F2Frec increases the recall by a factor of 8.8 in comparison with the centralized collaborative filter. A major reason behind this significant gain is the friendship establishment in F2Frec that relies on usefulness of users and friend networks. In contrast, centralized collaborative filter aggregates neighbors based on document ratings only. Unfortunately, this kind of similarity does not capture the contents of the documents.

VI. Related Work

In this section, we discuss the previous works that are most related to F2Frec. Previous works such as [3] and [5] exploit specific gossip protocols to aggregate the *neighbors* of each user. Then users use those *neighbors* (of *neighbors*) to serve their demands. However, these systems do not exploit users' social data and explicit friendship for recommendations.

In [6] and [8], users' social data (friends, trust, etc.) are exploited in computing recommendations. In [6], users' preferences are extracted from user's behaviors and asserted in users' FOAF files. Then the system aggregates users' FOAF files and clusters the users with similar preferences in one group. Then, when a user *u* in a group rates an item, the system determines whether the item is good enough to be recommended to other users in the group. However, aggregating users' FOAF files increases network traffic. In contrast, F2Frec lets each user maintain locally its FOAF file, and uses it to support its queries.

In [8], trust in users is also used as a basis for recommendations. The system lets each user *u* express its level of trust to each user it has interacted with. Then *u* measures the trust level between itself and each user in the network, and selects the most trustful as its neighbors. Those neighbors are used to compute the recommendations. However, inferring the trust relationships between users is time consuming and increases network traffic. In contrast, F2Frec computes the trust between indirect friends during query processing. Thus we do not need extra data and information to propagate and aggregate the trust network.

VII. Conclusion

In this paper, we proposed F2Frec, a P2P recommender system that leverages content and social-based recommendations by maintaining P2P social networks. The basic idea of F2Frec is to exploit the users' relevant topics of interest and friends' networks, in order to get high quality recommendations. F2Frec relies on gossip protocols to disseminate relevant users and their information, in order to let users establish friendship with new useful friends.

We use FOAF files to store users' friendship networks and their relevant topics of interest, and as a directory to redirect a query to the appropriate trustful and useful friends in a top-k approach.

In our experimental evaluation, using the TREC09 dataset and Wiki vote social network, we showed that F2Frec increases recall by a factor of 8.8, compared with centralized collaborative filter.

ACKNOWLEDGMENT

We would like to thank Bettina Kemme for her insightful discussions.

References

[1] J.-S., Breese, D., Hecherman, and C., Kadie, Empirical analysis of predictive algorithms for collaborative filtering. *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, 1998, pp. 43–52.

[2] D.-M., Blei, A.-Y., Ng, and M.-I., Jordan, Latent Dirichlet Allocation. *Journal of Machine Learning*, 2003, vol. 3, pp. 993–1022.

[3] F., Draidi, E., Pacitti, and B., Kemme, P2Prec: a P2P Recommendation System for Large-scale Data Sharing. *Tran. on Large-Scale Data- and Knowledge- Centered Systems, LNCS*, 2011, vol. 6790, No. 3, pp. 87-116.

[4] M., El Dick, E., Pacitti, R., Akbarinia, and B., Kemme, Building a peer-to-peer content distribution network with high performance, scalability and robustness. *Information Systems*, 2011, vol. 36, No. 2, pp. 222-247.

[5] A.-M., Kermarrec, V., Leroy, A., Moin, and C., Thraves, Application of Random Walks to Decentralized Recommender Systems. *OPODIS*, 2010, pp. 48–63.

[6] H.-J., Kim, J.-J., Jung, and G.-S., Jo, Conceptual framework for recommendation system based on distributed user ratings. *LNCS*, 2003, vol. 3032, pp. 115-122.

[7] L., Lacomme, Y., Demazeau, and V., Camps, Personalization of a trust network. *IEEE/ACM*, 2009, pp. 408–415.

[8] P., Massa and P., Avesani Trust-aware Collaborative Filtering for Recommender System. *LNCS*, 2004, vol. 3290, pp. 492-508.

[9] X.-H., Phan, October 2011, http://gibbslda.sourceforge.net

[10] S., Robertson and D.-A., Hull, The TREC-9 filtering track final report. *Proc. of 9th Text REtrieval Conf. (TREC-9)*, 2001, pp. 25-40.

[11] R., Sinha and K., Swearingen, Comparing Recommendation made by Online Systems and Friends. *Proc. of the DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*, 2001

[12] Wikipedia vote network, October 2011, http://snap.stanford.edu/data/wiki-Vote.html