Peer-to-Peer Recommendation for Large-Scale Online Communities

Fady Draidi

ACADÉMIE DE MONTPELLIER
U N I V E R S I T É **M** O N T P E L L I E R **I I**
- **S**CIENCES ET **T**ECHNIQUES DU **L**ANGUEDOC -

T H È S E

En vue de l'obtention du grade de
Docteur de l'Université de Montpellier II

Spécialité : **Informatique**
Formation Doctorale : **Informatique**
Ecole Doctorale : **Information, Structures, Systèmes**

Titre de la thèse

Recommandation Pair-à-Pair pour Communautés en Ligne à Grande Echelle

par
Fady DRAIDI

Soutenue publiquement le 09 Mars 2012

Section 27 - Informatique - devant le JURY composé de :

Mohand-Said Hacid	Professeur, Université Lyon I	Président
Sihem Amer-Yahia	DR CNRS, LIG, Grenoble	Rapporteur
Abdelkader Hameurlain	Professeur, Université Toulouse III	Rapporteur
Marie-Laure Mugnier	Professeur, Université Montpellier II	Examinateur
Esther Pacitti	Professeur, Université Montpellier II	Directeur de thèse
Patrick Valduriez	DR INRIA, LIRMM, Montpellier	Co-directeur de thèse

Recommandation Pair-à-Pair pour Communautés en Ligne à Grande Echelle

Résumé de la thèse

Les systèmes de recommandation (RS) et le pair-à-pair (P2) sont complémentaires pour faciliter le partage de données à grande échelle: RS pour filtrer et personnaliser les requêtes des utilisateurs, et P2P pour construire des systèmes de partage de données décentralisés à grande échelle. Cependant, il reste beaucoup de difficultés pour construire des RS efficaces dans une infrastructure P2P.

Dans cette thèse, nous considérons des communautés en ligne à grande échelle, où les utilisateurs notent les contenus qu'ils explorent et gardent dans leur espace de travail local les contenus de qualité pour leurs sujets d'intérêt. Notre objectif est de construire un P2P-RS efficace pour ce contexte. Nous exploitons les sujets d'intérêt des utilisateurs (extraits automatiquement des contenus et de leurs notes) et les données sociales (amitié et confiance) afin de construire et maintenir un overlay P2P social.

La thèse traite de plusieurs problèmes. D'abord, nous nous concentrons sur la conception d'un P2P-RS qui passe à l'échelle, appelé P2Prec, en combinant les approches de recommandation par filtrage collaboratif et par filtrage basé sur le contenu. Nous proposons alors de construire et maintenir un overlay P2P dynamique grâce à des protocoles de gossip. Nos résultats d'expérimentation montrent que P2Prec permet d'obtenir un bon rappel avec une charge de requêtes et un trafic réseau acceptables. Ensuite, nous considérons une infrastructure plus complexe afin de construire et maintenir un overlay P2P social, appelé F2Frec, qui exploite les relations sociales entre utilisateurs. Dans cette infrastructure, nous combinons les aspects filtrage par contenu et filtrage basé social, pour obtenir un P2P-RS qui fournit des résultats de qualité et fiables. A l'aide d'une évaluation de performances extensive, nous montrons que F2Frec améliore bien le rappel, ainsi que la confiance dans les résultats avec une surcharge acceptable. Enfin, nous décrivons notre prototype de P2P-RS que nous avons implémenté pour valider notre proposition basée sur P2Prec et F2Frec.

Mots-clés: Système pair-à-pair (P2P), système de recommandation (RS), communautés en ligne, réseaux sociaux, recherche d'information, gestion de données à grande échelle.

P2P Recommendation for Large-scale Online Communities

Thesis summary

Recommendation systems (RS) and P2P are both complementary in easing large-scale data sharing: RS to filter and personalize users' demands, and P2P to build decentralized large-scale data sharing systems. However, many challenges need to be overcome when building scalable, reliable and efficient RS atop P2P.

In this work, we focus on large-scale communities, where users rate the contents they explore, and store in their local workspace high quality content related to their topics of interest. Our goal then is to provide a novel and efficient P2P-RS for this context. We exploit users' topics of interest (automatically extracted from users' contents and ratings) and social data (friendship and trust) as parameters to construct and maintain a social P2P overlay, and generate recommendations.

The thesis addresses several related issues. First, we focus on the design of a scalable P2P-RS, called P2Prec, by leveraging collaborative- and content-based filtering recommendation approaches. We then propose the construction and maintenance of a P2P dynamic overlay using different gossip protocols. Our performance experimentation results show that P2Prec has the ability to get good recall with acceptable query processing load and network traffic. Second, we consider a more complex infrastructure in order to build and maintain a social P2P overlay, called F2Frec, which exploits social relationships between users. In this new infrastructure, we leverage content- and social-based filtering, in order to get a scalable P2P-RS that yields high quality and reliable recommendation results. Based on our extensive performance evaluation, we show that F2Frec increases recall, and the trust and confidence of the results with acceptable overhead. Finally, we describe our prototype of P2P-RS, which we developed to validate our proposal based on P2Prec and F2Frec.

Key-words: P2P system, recommendation system (RS), online communities, social networks, information retrieval, large-scale data management.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Motivations

Collaborative web 2.0 tools provide new opportunities for people to interact with each other within online communities, thus facilitating data, information and knowledge interchange, processing and publication. The most successful examples of online communities for publishing and locating information [62][6] are social networks (e.g., sites like MySpace [107] and Facebook [46]), wiki systems [164] (e.g., Wikipedia [164]), and content sharing web sites (e.g., sites like Citeulike [29] and Delicious [33]). Online communities have become very popular, and such popularity has translated into large amounts of data, content and knowledge being spread over very high numbers of users.

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 (within loops of activities), access terabytes of data, and generate terabytes of result data.

Therefore, with the constant progress in collaborative web 2.0 tools, combined with that of scientific observational instruments and simulation tools, the data overload keeps worsening and makes centralized data sharing difficult.

P2P networks have been successful at providing scalability, dynamicity and decentralized control. Thus, they can be used to build decentralized and scalable data sharing systems. Furthermore, the very nature of P2P with autonomous, collaborative participants (or peers) is well adapted to online communities. P2P is designed for direct sharing of participants' resources (processing power, storage capacity, network link capacity, data, etc.). These shared resources are accessible by other peers directly, without passing through intermediary entities, i.e., there is no central point of control. Therefore, the participating entities collaborate to perform tasks such as searching for other nodes, locating or caching content, routing requests, and retrieving content. P2P systems are fault-tolerant and scalable because they have no single point of failure.

P2P networks are characterized by their P2P overlay, on top of the physical network, which can be *unstructured*, *structured* or *dynamic*. Typically they differ on the

constraints imposed on how users are organized and where shared contents are placed [120]. The topology and degree of centralization of the P2P overlay are critical, because they have direct impact on the performance, reliability and scalability of the system. *Unstructured networks* are simple and incur low maintenance cost. However, this is at the expense of search techniques such as flooding, which incur much traffic consumption and hurt the scalability of the system. *Structured networks* provide an efficient, deterministic search that can locate content in a small number of hops. However, the maintenance of the structured overlay is complex and time consuming, which limits system scalability. *Dynamic overlays* have been proposed to address the limitations of both unstructured and structured networks, by providing self-organization of peers and contents in the overlay, with little overhead and network traffic. In a dynamic overlay, each peer continuously and dynamically constructs and updates potential contact peers of the overlay, using gossip protocols. Recent work on dynamic overlays [13][68][79][163] has also shown important performance gains.

P2P has been primarily used for file sharing, examples of popular systems being BitTorrent [18], eMule [44], Gnutella [51] and KaZaA [48][92]. Recently, P2P has also been applied to support high performance scientific workflow computing [109], and instant messaging [34]. P2P file-sharing systems have proven very efficient at locating content given specific queries [43]. However, popular P2P file-sharing systems such as Gnutella favour unstructured networks for their high flexibility, which are characterized by expensive search (flooding). Moreover, they only provide a very simple keyword search capability, trying to find the documents whose name or descriptions match the keywords provided by the user [82].

P2P systems have also been proposed for building decentralized search engines with more sophisticated P2P networks. These systems focus on enhancing search capabilities and reducing the network traffic consumed by search. To do so, information retrieval techniques such as clustering based on contents' semantics, establishing shortcut links to similar peers based on users' interests or social data, etc., are used to index, store, and organize data and peers. However, users may get overwhelmed not only with the high numbers of contents returned as results of their queries, but with ambiguous results (most of the results are not related to users' objectives expressed in their search). Therefore, it becomes hard for users to find the most valuable and relevant documents. Furthermore, these systems would typically return the same results for the same query submitted by two different users. Thus, the users' preferences such as interest in specific topics, past behaviors, rankings and ratings of contents they have explored, etc. is simply ignored.

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., to answer a 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.

In the mid-1990s, recommendation systems, or recommendation services (RSs), have been proposed to proactively deliver the right contents to the right users at the right time [2]. A RS returns to a user contents of interest, based on the contents' fea-

tures (content-based filtering), or based on the opinions given by other users on that contents (collaborative-based filtering). RSs enable users to provide feedbacks or ratings on the contents they explored. Then, based on this information and the contents' features, RSs can predict the preferences of users for yet unseen contents, and then suggest to users the contents that have the highest prediction relevance. RSs have become very popular and have been applied in many domains, including netnews [84], movies [59], musics [139], and jokes [55]. Furthermore, they have been deployed in many e-commerce applications, such as Amazon.com, NetFlix, Last.fm, MyStrands, iTunes, etc. to help users find products of interest (e.g., books, musics, videos) to purchase.

RSs that are based on users' ratings or features of the contents suffer from two problems: data sparsity, due to the fact that most users rate small number of contents; and cold start, due to the fact that a new user has not rated any content yet, or the new content has not been rated by any user yet. With the increasing popularity of web2.0 services such as social networks and collaborative tagging systems, social-based filtering has emerged. In social-based filtering, users' social data such as users' tags, friends and trusts, are used to enhance the performance of RSs, in particular, to increase the trust and confidence in the recommendation results [9][141][53]. For instance, Siham et al. [9] propose a generic system that can exploit users' information such as age, location, tags, bookmarks, etc. and the semantic of contents, in order to drive communities of interests, facilitate search, generate recommendations, explain the results, etc.

However, most RSs use a centralized infrastructure to manage and store users' preferences and contents, and process recommendations. These systems suffer from the single point of access and censorship problems, and require a heavy infrastructure (e.g., clusters) to provide scalability.

RS and P2P are both complementary in easing large-scale data sharing: RS to filter and personalize users' demands, and P2P to build decentralized large-scale data sharing systems. Both of them are trying to break through the limit of decentralizing and searching through huge amounts of data and users. However, many challenges need to be overcome when building scalable, reliable and efficient RS atop P2P.

In order to generate recommendations, RSs need to have the users' preferences (usually ratings) available. Unfortunately, users' preferences suffer from data sparsity and cold start problems, which may deteriorate quality (trust and confidence in the results) and reliability (in the sense that each user receives recommendations) of the system. Content-based filtering recommendation approach incorporates the features of the contents to enrich users' preferences and generate recommendations. Although they achieve good reliability, the recommendation results do not have good quality, primarily because social relations such as friends, trust, etc. are not incorporated in the recommendation process. The challenge is to leverage the features of contents with the users' feedbacks (ratings) and social data, dynamically extract users' preferences, and efficiently disseminate users' preferences in a decentralized infrastructure.

Recently, several research projects have proposed to use P2P in order to decentralize the RSs [156][106][80][133]. These proposals mostly use unstructured networks, and distribute users' ratings or social data over the peers in the network. When a user generates a recommendation, it first aggregates the users' preferences or social data from the network using flooding. Then, the user uses the aggregated information

to generate recommendations. However, aggregating users' preferences or social data through flooding incurs much traffic consumption and may deteriorate the scalability of the system. Obviously, much more work is needed in order to come up with scalable efficient P2P-RSs that provide high reliability and quality of recommendations.

1.2 Contributions

The goal of this thesis is to provide a novel and efficient decentralized RS for large-scale online communities. We exploit the use of users' topics of interest (automatically extracted from users' contents and ratings) and social data (friends and trust) as parameters to construct and maintain a social P2P overlay, and generate recommendations. In this work, we focus on large-scale communities, where users rate the contents they explore, and store in their local workspace high quality content related to their topics of interest.

This work has been carried out in the Zenith team (joint team between INRIA and University Montpellier 2, LIRMM, Montpellier) as part of the DataRing project (2009-2012), sponsored by the Agence Nationale de la Recherche (ANR), within the programme Future Networks and Services (VERSO). The DataRing project (http://www-sop.inria.fr/teams/zenith/dataring/), headed by Zenith, in collaboration with Leo (INRIA), LIG, LIRMM and Telecom Paristech addresses the problem of P2P data sharing for online communities, by offering a high-level network ring across distributed data source owners. The work produced in this thesis is the basis for DataRing's recommendation service.

Our work has evolved as follows. First, we have focused on the design of a scalable P2P-RS, by leveraging collaborative- and content-based filtering recommendation approaches. We have then proposed the construction and maintenance of a P2P dynamic overlay using different gossip protocols. Second, we have moved to a more complex infrastructure in order to build and maintain a social P2P overlay (that exploits social relationships between users). In this new infrastructure, we leverage content- and social-based filtering, in order to yield a scalable P2P-RS that has high quality and reliable recommendation results. Finally, we have implemented a prototype of our proposal as an application for the Shared-Data Overlay Network (SON), an open source development platform for P2P networks developed in the Zenith team.

In this thesis, we make the following contributions.

First, we survey the related work. We introduce the main concepts and approaches of recommendation systems, and their limitations. We introduce the three main classes of P2P systems (unstructured, structured and dynamic) and show the advantages of dynamic systems for our work. We also highlight the requirements that are needed to design P2P recommendation systems. Then we review the existing solutions related to information retrieval, called *P2P content management systems*, and P2P recommendation systems that are based on users' preferences, called *P2P prediction systems*.

Our second contribution consists in building a P2P recommendation system, called P2Prec [36][38], which facilitates document sharing for on-line communities.

P2Prec leverages content- and collaborative-based filtering recommendation approaches. It uses users' relevant topics of interest and gossip exchanges, to organize users and serve queries. P2Prec adopts a dynamic P2P overlay that is constructed and maintained through gossip protocols. A user's relevant topics of interest are automatically computed by analyzing the documents the user holds. The relevant users are used to serve queries, and a user is considered relevant in a topic if it is interested in this topic, and holds a sufficient number of highly rated documents on that topic. Accordingly, the user can provide recommendations for that topic. P2Prec uses semantic-based gossip protocols to efficiently disseminate information about users' topics and relevant users, and gossip views as a directory to serve users' queries. Semantic-based gossip protocols allow each user to selectively maximize the number of relevant users at its gossip view, given that those relevant users are similar to the user and can serve its demands. In addition, P2Prec uses an efficient query routing algorithm that selects the best relevant users to recommend documents based on the gossip view and query topics. Our simulation results show that P2Prec has the ability to get reasonable recall with acceptable query processing load and network traffic.

Our third contribution aims at extending P2Prec with the social relationships between users as a parameter for recommendation, in order to increase the trust and confidence of recommendation. Thus, we propose F2Frec [41], which leverages content- and social-based filtering recommendation approaches, in order to construct and maintain a P2P and friend-to-friend network, and to facilitate recommendations. F2Frec uses new metrics based on users' relevant topics of interest, and similarity (among users and their respective friend network), in order to enable friendship establishment and to facilitate recommendations. Given that a gossip protocol is used to disseminate users' relevant topics of interest, in order to enable users find new interesting friends. F2Frec stores a user's friends along with their information in a specific file, which is also used as a directory to serve user queries. F2Frec uses an efficient query routing algorithm. Moreover, we propose to rank the returned recommendations by taking into account the semantic similarities, content popularity, distance and trust between the query's initiator and responders. Based on our extensive performance evaluation, we show that our approach increases recall, and the trust and confidence of the results with acceptable overhead.

Our fourth contribution is the development of a P2P-RS prototype, which is based on P2Prec and F2Frec, as open source software (http://www-sop.inria.fr/teams/zenith/p2prec/). The prototype is developed as an application on top of the Shared-data Overlay Network (http://wwww-sop.inria.fr/teams/zenith/SON), an open source development platform for P2P networks developed in Zenith. We choose SON because it makes easy the development of a P2P application: the developer only writes the code logic for the behaviors of the peer (as in a simulator). The complex aspects of asynchronous messages between peers are automatically generated and managed by SON, i.e., the developer does not deal with complex distributed programming aspects. We built a full-fledge demonstration of this prototype using the Ohsumed documents corpus [60], showing how friendship establishment, query processing, gossip protocol, etc. are involved.

To summarize this thesis has produced:

- Two journals papers:

1. F. Draidi, E. Pacitti, B. Kemme. P2Prec: a P2P Recommendation System for Large-scale Data Sharing. Transaction on Large-Scale Data- and Knowledge- Centered Systems, LNCS, 6790(3), 87-116, 2011.
2. A. Bonifati, G. Summa, E. Pacitti, F. Draidi. Semantic Query Reformulation in Social PDMS. CoRR abs/1111.6084, 2011. Submitted on 25 Nov 2011 to Data & Knowledge Engineering (DKE) Journal.

- Two conferences papers:
  1. F. Draidi, E. Pacitti, P. Valduriez, B. Kemme. P2Prec: a Recommendation Service for P2P Content Sharing Systems. Bases de Donnees Avancees (BDA), 26, 21-40, 2010.
  2. F. Draidi E. Pacitti, M. Cart H-L. Bouziane. Leveraging Social and Content-based Recommendation in P2P Systems. The 3rd Int. Conf. on Advances in P2P Systems (AP2PS), 13-18, 2011.
- Two demo papers:
  1. F. Draidi E. Pacitti, D. Parigot G. Verger. Demo of P2Prec: a Social-based P2P Recommendation System. Journées Bases de Donnees Avancées (BDA), 27, 5-8, 2011.
  2. F. Draidi E. Pacitti, D. Parigot G. Verger. P2Prec: a Social-based P2P Recommendation System. Proceedings of the 20th ACM Conf. on Information and Knowledge Management (CIKM), 2593-2596, 2011.
- Two deliverables of the DataRing project:
  1. F. Draidi, E. Pacitti, P. Valduriez. Deliverable D5.2: demo of replication, caching and indexing services. DataRing Project, Dec. 2010.
  2. F. Draidi, E. Pacitti, P. Valduriez. Deliverable D5.3: replication, caching and indexing services - experiments report. DataRing Project, Dec. 2011.
- One prototype delivered as open source software: http://www-sop.inria.fr/teams/zenith/p2prec/

1.3 Thesis Organization

The rest of this thesis is organized as follows.

In Chapter 2, we provide a literature review of the state-of-the-art in P2P-RSs. First, we introduce the main concepts and approaches of recommendation systems, and their limitations. Then, we introduce P2P systems, identify their main classes, and highlight the requirements that are needed to design P2P-RSs. Finally, we review the existing solutions in P2P content management systems and P2P prediction systems.

Chapter 3 presents P2Prec, our proposed P2P recommendation system that leverages content- and collaborative-based filtering recommendation approaches. After the problem definition, we introduce P2Prec basic concepts such as topics of interest and relevant users. Then, we explain how the P2Prec overlay is constructed and maintained via gossip protocols, and describe new semantic-based gossip protocols. Next,

we describe our solution for query routing and recommendation ranking. We conclude with our extensive experimental evaluation and results.

Chapter 4 extends P2Prec in order to enhance the confidence and trust of recommendations. After giving the general overview of F2Frec and the problem definition, we introduce F2Frec basic concepts, and present our social metrics and how we manage friendship establishment. Then, we describe the solution for retrieving and ranking recommendations. Next, we explain how to manage the dynamicity of users' topics of interest. Finally, we present our experimental validation.

In Chapter 5, we describe the design and implementation prototype of our proposal. First, we give a detailed description of the architecture and implementation of our proposal. Then, we describe the demonstration of our prototype using a real data set.

Chapter 6 concludes and highlights future directions of research.

Chapter 2 State-of-the-Art

Abstract. In order to define the problems we address in this thesis, this chapter reviews the state-of-the-art in P2P recommendation systems. First, we describe centralized recommendation systems, their approaches, and limitations. Second, we discuss P2P systems as an alternative, scalable solution for decentralized recommendation systems, and extract their requirements. Finally, we evaluate the existing P2P content management systems and P2P prediction systems.

2.1 Introduction

RSs help filtering out the contents the users may like to explore from huge amounts of contents based on what they like. RSs exploit the users' preferences (interest, expertise, friends, ratings, etc.) and suggest contents or information items (e.g., movies, documents, Web pages, CDs, or books) of interest to users according to their preferences [15][54]. Basically, RSs analyze users' historical patterns (ratings, purchasing, etc.) to find and recommend new contents that the users might be interested in and like to explore or purchase [125]. Notice that a user's historical patterns, called *user profile, user model, user preferences,* may include information about user (such as name, age, location, etc.), or user's interaction history (what the user bought, viewed, rated, etc.), or description of the items (item's features) the user has seen, purchased, explored or rated.

Collaborative Web2.0 tools such as social networks have become very popular and make it now very easy to publish users' social data. Social networks allow anybody to create their own online profile, allow friends to join in, and communicate with them. Typically, a user's profile includes personal information such as name, age, location, topics of interest, expertise, etc. The emergence of Web2.0 and the growing popularity of online social networks have encouraged exploiting users' social data in recommendation systems to improve the quality of recommendations [9][141].

Most recommendation systems for web data are implemented in a centralized infrastructure provided by a single operator (e.g. Google or Yahoo). We classify RSs between content-based filtering [23][116], collaborative-based filtering [25][137], and social-based filtering [85][88][140]. Content-based filtering recommends to a user *u* items that are similar to *u*'s previously rated items. Collaborative-based filtering, in contrast, recommends to *u* items that have been rated by users who share similar in-

terests based on rating behavior. Social-based filtering recommends to *u* items that have been rated by its friends, or trustful users.

Typically, in a centralized infrastructure, all users and their rated contents, denoted by *user-item* matrix, are stored in a central server or by a single operator, which also performs all the recommendation processes. Centralized infrastructure might be not applicable for large-scale online communities due to several reasons:

1. Single operators are expensive and might be not affordable especially to non-profit online communities.
2. Aggregating all users and their data, and stores them on a single server or single operator is not an easy task, and it consumes a lot of time and network traffic.
3. Storing the users and their data, and generating recommendations on a single server or single operator leads to a single point of access, and this may deteriorate the availability of the recommendation service.
4. More importantly, many participants of online communities are reluctant to give full control over their private data to a provider who can sell it to other businesses and worse, leave such control to third parties of unknown affiliations.

P2P recommendation for web data based on collaborative-based filtering has been recently proposed [79] with promising results. However, designing scalable, reliable and efficient P2P recommendation system that leverages users' social data and preferences arises very interesting challenges.

Notice that throughout this chapter, we use the terms "contents", and "items" interchangeably, based on context, to refer to the service that a recommendation system is designed to recommend. Given that an item or content can be a document, movie, website, book, CD, an image, etc.

In this thesis, we exploit techniques from two areas: information retrieval and recommendation systems. Thus, we review the literature about existing solutions in both areas in the context of P2P systems.

The rest of this chapter is organized as follows. Section 2.2 introduces the main concepts and approaches of recommendation systems, and their limitations. Section 2.3 introduces P2P systems and identifies their main classes. We also highlight the requirements that are needed to design P2P recommendation systems. Section 2.4 reviews the existing solutions related to information retrieval, called *P2P content management systems*. Section 2.5 reviews the existing solutions for P2P recommendation systems that are based on users' preferences, called *P2P prediction systems*. Section 2.6 summarizes our observations regarding the state-of-the-art related to the thesis. Section 2.7 concludes.

2.2 Recommendation Systems

In this section, we introduce the main concepts and approaches of recommendation systems, and their limitations. We start with an overview that helps classifying recommendation systems between collaborative-based filtering, content-based filtering and social-based filtering. Then we describe each class of recommendation approach with their limitations.

2.2.1 Overview

The explosive growth of web-scale collaboration has increased the amount of information that is available to users. Therefore, people use a variety of strategies to search for contents and make choices about what to explore. Recommendation systems have emerged as software applications that help users with options to consider and explore. Recommendation systems have roots back to *information retrieval* [132]. Information retrieval deals with searching for the contents that match a given query, and then return and retrieve those contents to the users. These systems return all the contents that are related to a given query. Thus, users get overwhelmed with the high numbers of contents returned as results of their queries, and it becomes hard for them to find the most valuable and relevant contents. Information filtering [159] has emerged to overcome this problem, by taking into account users' historical patterns, in order to identify the contents that the users might be interested in. One of the most successful and popular class of information filtering is recommendation systems.

Recommendation is ubiquitous in our daily life, where we must choose between alternatives based on opinions and advice 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 advice, and benefit from each other's experience without human intervention, recommendation systems have emerged. Since the first work carried out by Goldberg et al [54], RSs have been used in major applications such as e-commerce, e.g. Amazon.com, Netflix.com or CDNow [135].

Figure 2.1 gives a general overview of a recommendation system and its main components. In general, a RS first collects a user's historical patterns they have expressed, either explicitly or implicitly [11][49]. Then it finds other users with similar patterns or items in the historical patterns. Finally, the RS uses the data from those similar users or items to suggest items the user might be interested in.

To avoid the drawbacks of *collaborative-*, *content-*, *and social-based filtering* recommendation approaches and exploit synergetic effects, combinations of them have been developed in the so-called *hybrid filtering* [136]. Since *hybrid filtering* is a combination between *collaborative-*, *content-*, *and social-based filtering*, in the following, we focus on the three main approaches of RSs.

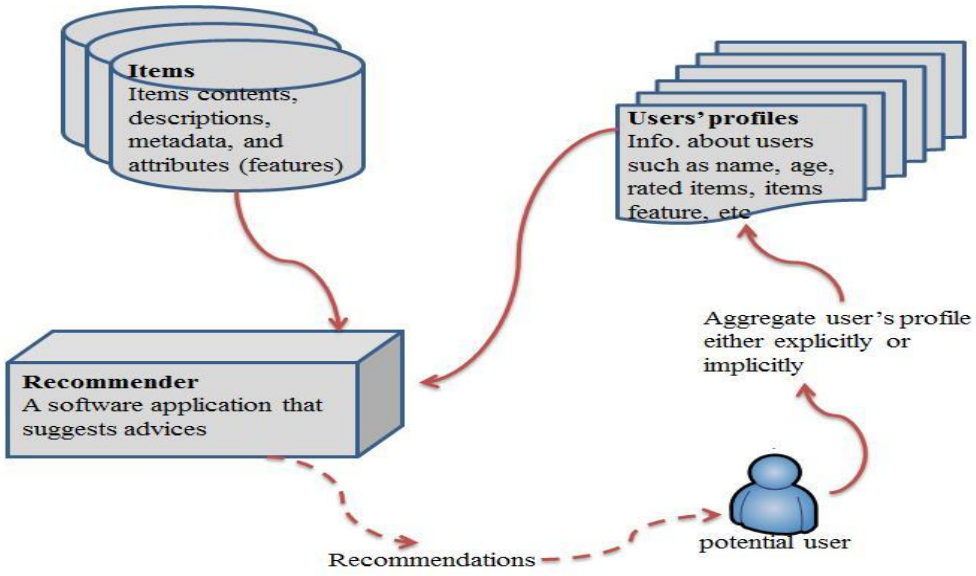


Figure 2.1. Overview of recommendation systems

2.2.2 Collaborative-based Filtering

Collaborative-based filtering is one of the most popular classes of recommendation systems [102][170]. Collaborative-based filtering tries to automate the process of "word-of-mouth", people who have the same preferences and most probably have similar taste and interest. In collaborative-based filtering, users express their preferences by rating items either explicitly or implicitly [104][108]. The user ratings, either explicit or implicit, are often represented by discrete values within a certain range, e.g., between 1 and 5.

Collaborative-based filtering typically works with three generic steps [139] (see Figure 2.2): (1) measure the similarity between a user *u* (the user asking for recommendations) and all users in the system; (2) select those users who are most similar to *u*, denoted by *neighbors(u)*; (3) normalize and compute the weighted sum of the *neighbors(u)* ratings, then make suggestions based on those ratings. Collaborative-based filtering has been widely used for building RSs, e.g. in GroupLens [124], Ringo/Firefly [139], Tapestry RS [54] and Recommendation [61].

More formally, in collaborative-based filtering users' preferences or profiles are modeled in a *UxI* user-item matrix *R*, as shown in Figure 2.2, where *U* represents the set of users, *I* represents the set of items in the system, *n* is the number of users and *m* is the number of items. Each entry $r_{u,i}$ of *R* includes the rating given by user *u* for item *i*, where $r_{u,i} = r$ indicates that user *u* rated item *i* by a value of *r*, and $r_{u,i} = \Phi$ indicates that user *u* has not rated item *i* yet. Each row $r_u \in R$ corresponds to a user's profile and includes *u*'s items rating. The goal of the collaborative-based filtering system is to predict missing entries in matrix *R*.

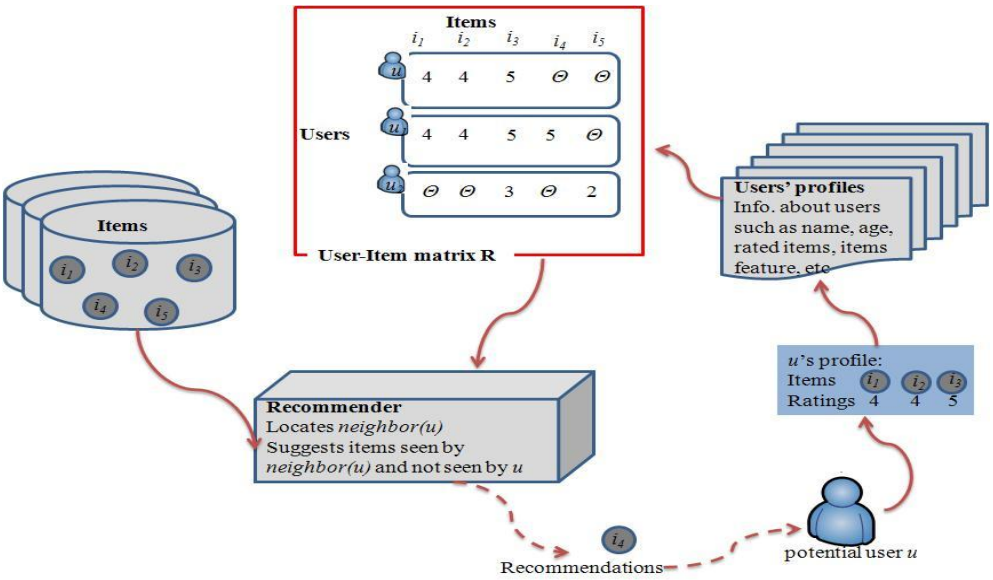


Figure 2.2. Collaborative-based filtering

In order to form the neighbors of each user *u*, a similarity measure is used to compute the similarity between *u* and all the users in the system. One popular measure for this is *cosine similarity* [92][136]. The similarity between a user *u* and another user *v*, denoted by *sim(u,v)*, is computed by summing the products between the ratings that have been given by *u* and *v* over their items:

$$sim(u,v) = \frac{\sum_{i \in I} r_{u,i} * r_{v,i}}{\sqrt[2]{\sum_{i \in I} r_{u,i} * r_{u,i}} * \sqrt[2]{\sum_{i \in I} r_{v,i} * r_{v,i}}} \quad (2.1)$$

Another way to compute similarity between users is to use the Pearson correlation coefficient [100][162]. It was first introduced into collaborative-based filtering as a weighting method in the GroupLens project [124]:

$$sim(u,v) = \frac{\sum_{i \in I} (r_{u,i} - \overline{r_u}) * (r_{v,i} - \overline{r_v})}{\sqrt[2]{\sum_{i \in I} (r_{u,i} - \overline{r_u}) * (r_{u,i} - \overline{r_u})} * \sqrt[2]{\sum_{i \in I} (r_{v,i} - \overline{r_v}) * (r_{v,i} - \overline{r_v})}} \quad (2.2)$$

where r_u is the average ratings of *u*. Once the similarity between *u* and *v* has been computed, a neighborhood is formed by using the Best-n neighbors [124] or Similarity-thresholding [139]. With the Best-n technique, the best *n* similar users are selected as neighbors for *u*. With the similarity-thresholding technique, a user *v* is considered a neighbor to *u* if *sim(u,v)* exceeds the system-defined threshold.

The last step of the collaborative-based filtering process is to generate the recommendations list for *u* from its neighbors. In this step, the final rating prediction of each item $i \in I$ that user *u* did not explore yet, denoted by $r_{u,i}^*$, is predicted by normalizing and computing the weighted sum of the *neighbors(u)* ratings on item *i* :

$$r_{u,i}^* = \frac{\sum_{v \in neighbour(u)} r_{v,i} * sim(u,v)}{\sum_{v \in neighbour(u)} sim(u,v)} \quad (2.3)$$

Then item *i* is added to a prediction list, denoted by *predictList*, a list that includes the items that *u* did not rated yet, along with their rating predictions. After that, the recommendations list is generated from the *predictList* using the top-k or thresholding method. With the top-k method, collaborative-based filtering ranks each item $i \in predictList$ based on its rating prediction $r_{u,i}^{*}$. Then, collaborative-based filtering selects the top-k items and adds them to the recommendation list. With the thresholding method, collaborative-based filtering selects each item $i \in predictList$ such that its rating prediction $r_{u,i}^{*}$ exceeds the system-defined threshold, and adds it to the recommendation list.

Let *N* be the number of users in the system, *M* the number of items in the system, and *K* the number of neighbors of each user. Collaborative-based filtering performs $O(N^2xMxK)$ calculations to measure the similarity between users and construct their neighbors, and performs $O(K)$ calculations to predict a rating for an item [27]. In addition, it needs $O(NxM + NxK)$ of storage space, to store the user-item matrix and users neighbors, excluding the storage space needed to store the items' contents.

For instance, using a single server to construct users' neighbors for the MovieLens [58] dataset, which holds 10 million ratings for 10000 movies rated by 72000 users, selecting the top similar 100 users as the neighbors for each user requires $5184x10^{12}$ calculations. If we consider that each calculation takes 1 Nano second to be computed on that server, we need 60 days to finish constructing users' neighbors. If we consider that the length of each rating value is 2 bytes, the server consumes 1.355 Gigabytes to store the user-item matrix only, without considering the storage required to store the movies' contents.

Infrastructures such as clusters, grids, and cloud computing might be used to decrease the computation time and storage space required to perform recommendations. However, these infrastructures are somehow expensive and significant time and costs should be invested, to scale-up and cop with the increasing number of users and items.

2.2.2.1 Limitations

Collaborative-based filtering is the widely successful recommendation approach, and has become used in many e-commerce systems. However, collaborative-based filtering suffers from several drawbacks:

- **Data sparsity**: Most users rate small numbers of items in the system [92], thus those users might not find similar users.
- **Cold start problem**: This is referring to the problems occurring with a new item or a new user [138]. A new user who has not rated any items yet will not find similar users to help in finding recommendations [2][168]. On the other hand, when a new item is introduced into the system, and no user has rated that item yet, it is not possible to recommend that item in any way.
- **Limited Scalability**: this is referring to the problem of resource consumption. Storing users' profiles in one place consumes much storage that must increase tremendously as the number of users and items increase. Moreover, measuring the similarity between users is time consuming, and it increases exponentially as the numbers of items and users increase. This is a major concern for e-commerce web sites providing a lot of recommendations while serving millions of users.

2.2.3 Content-based Filtering

Unlike collaborative-based filtering, content-based filtering works by suggesting to the user items that are similar to items that the user has seen or rated [23]. In Figure 2.3, the circles represent the items in the system, where the items with the same colours are similar items. In content-based filtering, the similarity measure is computed between the items the user has seen or rated and the items that the user did not see or rate yet. Items with high similarity are suggested to the user.

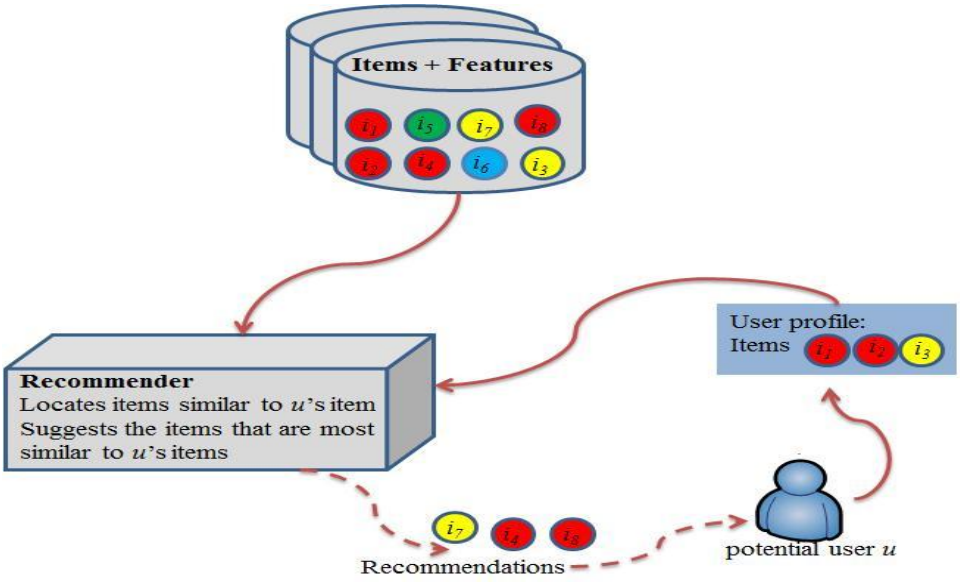


Figure 2.3. Content-based filtering

To measure the similarity between items, each item is identified by a set of features and attributes that are usually extracted from its content or description. Therefore, content-based filtering is designed mostly to recommend text-based items or items that have text descriptions. Usually the description or content of items in these systems is described and represented with keywords. For instance, the Fab system [15], a web page recommendation, represents the web page preferred by a user with the 100 most important keywords. Similarly, the Webert system [116] represents a document preferred by a user with the 128 most informative words.

More formally, each item $i \in I$ is represented by a vector of keywords, denoted by $V_i = \{ w_{i,1}, w_{i,2}, .., w_{i,j}, .., w_{i,k} \}$, where I represents the set of items in the system, $w_{i,j}$ is the weight (importance) of keyword j in item i, and $k \in K$ is the number of unique keywords in the system, where K is the set of the unique keyword in the system. One of the popular metric used to compute the weight w_{ij} of a keyword j in an item i is the normalized TF-IDF metric [12]:

$$w_{i,j} = \frac{f_{i,j}}{max\, f_{i,z}} * log \frac{N}{n_i} \qquad (2.4)$$

where $f_{i,j}$ is the number of times the keyword j appears in item i, max $f_{i,z}$ is the number of times the most frequent word z appears in item i, n_i is the number of items that have the keyword j, and N is the total number of items in the system.

Once the vectors of keywords of a user u's items have been constructed, we add them to u's profile. Accordingly, a user u's profile consists of vectors of keywords of the items that u has rated. In more details, a user u's profile can be represented as a matrix P with dimension $|I_u|xk$, where $I_u \subset I$ is the set of items u has rated, and $|I_u|$ is the cardinality of the set of items u has rated. Each row $P_i \in P$ corresponds to an item i's vector of keywords such that $i \in I_u$, and includes the weight of the item i's keywords.

Once a user u's preferences or profile has been constructed, we measure the similarity between each item $i \in I_u$ and each item $j \in I$ that user u did not see or rate yet. There are several techniques to evaluate the similarity between a user profile and an item using: Boolean methods [90][160], vector-space methods [131], probabilistic models [126], neural networks [81][166], fuzzy set models [110], etc. One popular technique is vector-space that uses the *cosine similarity* measure. The similarity between an item i and another item j, denoted by *sim(i,j)*, is computed by multiplying the components of the two items keywords vectors, and then summing those products. The result of sum is normalized by dividing it over the product of the lengths of the two items keywords vectors:

$$sim(i,j) = \frac{\sum_{k \in K} w_{i,k} * w_{j,k}}{\sqrt[2]{\sum_{k \in K} w_{i,k} * w_{i,k}} * \sqrt[2]{\sum_{i \in K} w_{j,k} * w_{j,k}}} \quad (2.5)$$

After similarity processing, recommendations are generated and provided to user u by selecting the m items that are most similar to the items that u has seen and rated. Alternatively, the items of which rating predictions exceed a system-defined threshold can be recommended to the user.

Let N be the number of users in the system, M the number of items, and K the number of similar items that are chosen for each item. Content-based filtering performs $O(M^2xNxK)$ calculations to measure the similarity between items and select their similar items, and $O(K)$ calculations to predict a rating for an item [27]. In addition, it needs $O(NxM + MxK)$ of storage space. The example of MovieLens dataset requires $72x10^{11}$ calculations to construct items similarity using a single server. The same if we consider that each operation takes 1 Nano second to be computed on that server, we need 2 days to finish the similarity process.

2.2.3.1 Limitations

Content-based filtering alleviates the problems of data sparsity [3][17] and new item. Similarity is based on item contents or descriptions, so the user who rates few items has the opportunity to receive recommendations, and the item that has been never rated yet has the opportunity to be recommended to users. However, content-based filtering has the following problems:

- **Overspecialization**: a user is limited to receive items that are only similar to the items it has seen or rated, and thus might not explore new interesting topics. Abbassi et al. [1] propose Outside-The-Box (OTB) recommendation. OTB groups

similar items in boxes, and gives each user *u* a chance to receive recommendation items from boxes that include items not similar to *u*'s items.

- **New user problem**: the user who did not have past preferences, and did not rate or see items yet, will not receive recommendations from the system

2.2.4 Social-based Filtering

The emergence of Web2.0 provides opportunities for people to interact with each other, and facilitate knowledge interchange, processing and publication. Online communities such as social networks (e.g. sites like Facebook and MySpace, the Friend-Of-A-Friend (FOAF) project [152], etc.), and wiki systems [164] (e.g. Wikipedia) are some of the most successful examples of Web2.0 services for publishing and locating information [62][6]. The advantages of mass collaboration such as faster production and better accuracy of knowledge and data can also be brought to all kinds of companies and, for instance, help them create better services and items faster at lower cost.

Online social network is a very popular service that exploits Web2.0 technology. Social network allows anybody to present themselves through their online profile, and create, edit, annotate and share data with other users. Independent of the content, the user maintains links to other users, which indicates trust, friendship or shared interest. Social network is typically modeled as a graph, where nodes represent users, and an edge between two nodes refers to the relationship between two users. In practice, an edge can refer to any type of relationships e.g., family, trust, friends, common interest, like minded, etc. Moreover, a numeric value (e.g., between 0 and 1) can be attached to an edge between two users, to represent the strength of the relationship between the two users. Social network exhibits the *small-world phenomena* [144][105], that is, a user *u* can contact any other user *v* in the system in few hops. Therefore, the social network structure gives users ability to find new users with similar interests, and locate content in an efficient way.

Social-based filtering leverages social network links and data to improve the quality of search and recommendation results [9][85]. Unlike collaborative-based filtering or content-based filtering, social-based filtering does need to measure the similarity between users or items. In contrast, social-based filtering follows the relationships between users to generate recommendations. Given that the relationships between users are extracted and defined before recommendation process is started.

More formally, social-based filtering is modeled as a graph $G=(U,E)$, where U is the set of users in the system, and E is the set of edges in the system such that there is an edge $e(u,v)$ if there is a direct relationship between users u and v. For ease of explanation, we use the friendship to represent the relationship between users. Thus, we consider that there is an edge $e(u,v)$ if users u and v are direct friends, and the numeric value attached to the edge, denoted by $w_{u,v}$, is the strength of the friendship between u and v. In the example of Figure 2.4, users u and v are direct friends, and the strength of the friendship between u and v is 0.8.

Each user $u \in U$ has a set of direct friends, denoted by $friend(u)$, and a set of items $I_u \subset I$ the items u has rated, where I is the set of items in the system. Given that u has

rated each item $i \in I_u$ by a value $r_{u,i}$. The goal of social-based filtering is to predict the rating value $r_{u,i}^*$ for each item $i \in I$ and $i \notin I_u$. To do so, social-based filtering computes the rating prediction $r_{u,i}^*$ by normalizing and computing the weighted sum of the *friend(u)* ratings on item *i* :

$$r_{u,i}^* = \frac{\sum_{v \in friend(u)} r_{v,i} * w_{u,v}}{\sum_{v \in friend(u)} w_{u,v}} \quad (2.6)$$

Then the top-k items that have the highest rating predictions are recommended to the user *u*. Alternatively, the item *i* that has rating prediction $r_{u,i}^*$ exceeds the system-defined threshold can be recommended to the user *u*.

The set of users used to compute the rating prediction of an item *i* is not limited only to *friend(u)*, and indirect friends (friend of friends) can be included to compute the rating prediction of the item *i*. In that case, propagation methods such as averaging, shortest path, etc. are used to compute the strength of the friendship between indirect friends.

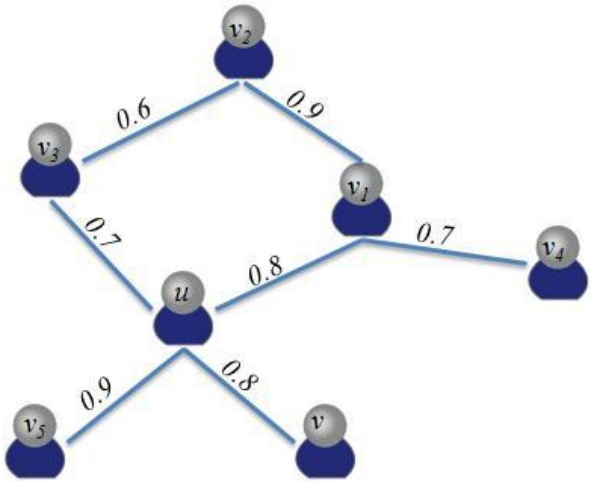


Figure 2.4. A snapshot of social-based filtering graph

Several social relationships are leveraged in RSs including friendship [85], trust [88], tags [155][140], etc. For instance, the authors of [88] propose to select the neighbors of each user *u* based on the trust network, in order to overcome the cold start problem. Each user *u* computes the trust value between itself and each user in the system. Then, the top-k trustful users are chosen as the neighbors of *u*. In [155][140], the authors exploit users' tags to enhance recommendation. A tag is a metadata assigned by users to items they have explored and shared, in order to annotate and categorize those items. Users' tags are used to measure the similarity between users and tags' semantics to identify the similarity between items.

Kruk et al. [85] introduce an approach for "semantic social collaborative-based filtering" based on FOAF files. Each user *u* stores its interests and friends in its FOAF file. The system aggregates users' FOAF files, and uses users' explicit social net-

works to link each user *u* with friends and friend of friends that their interests are similar to *u*'s interests.

Siham et al. [9] propose a generic approach, called community-driven information exploration in order to build communities, facilitate search and recommendations, etc. This approach relies on SocialScope [8], an architecture for aggregating data from content and social web sites, and Jelly [7], a language that provides primitives (rating, tagging, users, and items) that can be used to find the relations between users and/or contents. Based on SocialScope's architecture and Jelly primitives, users can derive and generate topics of interest, community, recommendations, and grouping, ranking and explaining results.

To conclude, exploiting social relations such as friends, trust, etc. enhances the quality and performance of RSs, and alleviates the problems of sparsity and new user. In addition, it increases the confidence and trust in the recommendation results, because they are returned from trusted friends.

2.3 P2P Systems

P2P systems have gained popularity to share users' resources (i.e. processing power, files, information, etc.). These shared resources are accessible by other peers directly, without requiring intermediary entities, i.e. there is no central point of control. P2P systems are built on top of the physical network (typically the Internet), and thus also referred to as *P2P overlay networks* (or P2P networks for short) and each peer is connected to a set of peers, called *neighbors*. P2P networks rely on a *topology* that defines how peers are connected, and a *routing* and *searching protocol* that defines how peers locate and search for contents and/or other peers.

P2P systems provide scalability, fault tolerance, and self-organization without requiring a dedicated infrastructure. Thus, they are a potential solution for building large-scale file and content management sharing systems at low cost [91][134]. P2P techniques have become very popular and being used in different contexts. Although, they are very dynamic, they give users the ability to join and leave the network at any time. Therefore, P2P networks should have the ability of self-organization once peers join and leave, and be scalable to any growing number of peers.

We classify P2P networks according to their degree of centralization and their topology between *unstructured*, *structured* and *dynamic* networks. Typically they differ on the constraints imposed on how users are organized and where shared contents are placed [120]. The topology and degree of centralization are critical, because they have direct impact on the performance, reliability and scalability of the system. Note that our classification of P2P networks differs a bit from that in [113] (which distinguishes between unstructured, structured and hybrid) in order to better fit the context of this thesis.

In the rest of this section, we give a general overview of these classes of P2P networks and their main search techniques. We also highlight the requirements that are needed to design P2P recommendation systems, which are at the heart of this thesis.

2.3.1 Unstructured P2P Networks

Unstructured P2P networks impose no restriction on data placement in the overlay topology [122]. The overlay network is created in a nondeterministic (ad hoc) manner and the data placement is completely unrelated to the overlay topology (see Figure 2.5). Once the overlay is constructed, each peer knows its neighbors, but does not know the resources that they have.

Unstructured networks are the earliest examples of P2P systems whose core functionality remains file sharing. In these systems replicated copies of popular files are shared among peers, without the need to download them from a centralized server. Examples of these systems are Gnutella, KaZaA, and BitTorrent.

A fundamental issue in all P2P networks is the type of index to the resources that each peer holds, since this determines how resources are searched. Unstructuted networks typically use a distributed index, where each peer maintains metadata for resources that it holds.

Unstructured networks typically use flooding protocols to disseminate discovery messages or queries [120]. Flooding is a widely used in P2P file sharing applications [51] due to its simplicity. With flooding, each query is attached with a *Time-To-Live* (TTL) value. When a peer *p* issues a query *q*, it forwards *q* to all its neighbors, which in turn forward *q* to their neighbors until TTL becomes zero. However, flooding consumes a lot of network bandwidth, and thus may hurt scalability. Furthermore, TTL restricts the number of nodes that are reachable.

There have been approaches to address the problems of flooding. A straightforward method is for each peer to choose a subset of its neighbors and forward the request only to those. How this subset can be determined may vary. For example, the concept of random walks can be used [96] where each peer chooses a neighbor at random and propagates the request only to it. Each visited neighbor periodically contacts the query originator, asking whether the query was satisfied or not. The main disadvantage of random walk is its highly variable performance, because success rates and the number of found answers vary greatly depending on the network topology and the random choices.

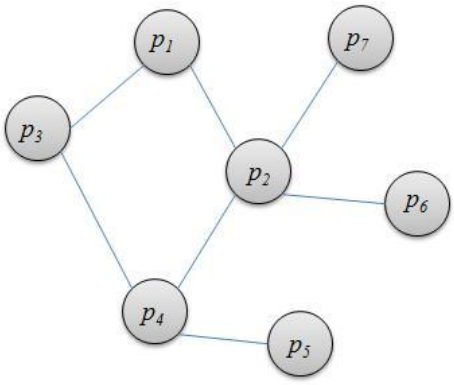


Figure 2.5. Example of unstructured P2P network

2.3.2 Structured P2P Networks

Structured P2P networks have emerged to address the scalability issues faced by unstructured P2P networks, by tightly controlling the overlay topology and the placement of resources. Thus, they achieve higher scalability at the expense of lower autonomy as each peer that joins the network allows its resources to be placed on the network based on the particular control method that is used.

Structured networks exploit a distributed data structure such as a tree or hash table to control content placement and location, and thus, provide efficient, deterministic search that can locate data in a small number of hops. Popular structured P2P systems are Oceanstore [86], CAN [123], Pastry [128], CHORD [148] and Tapestry [169]). The most popular form of structured network is the Distributed Hash Table (DHT), which implements distributed hashing [122] (see Figure 2.6).

The most popular indexing and data location mechanism that is used in structured P2P networks is dynamic hash table (DHT). DHT-based systems provide two API's: *put(key, data)* and *get(key),* where key is an object identifier. The key is hashed to generate a peer id, which stores the data corresponding to object contents. Given a search query based on a given key *k,* a DHT can lookup the peer that stores the data for *k* efficiently, usually in *O(log n)* routing hops where *n* is the number of peers.

Structured P2P networks provide basic techniques for routing queries to relevant peers and this is sufficient for supporting simple, exact-match queries. For instance, as noted earlier, a DHT provides a basic mechanism to efficiently look up data based on a key value. However, supporting more complex queries such as top-k, join and range queries is more difficult and has been the subject of much recent research (see Chapter 16 in [113]).

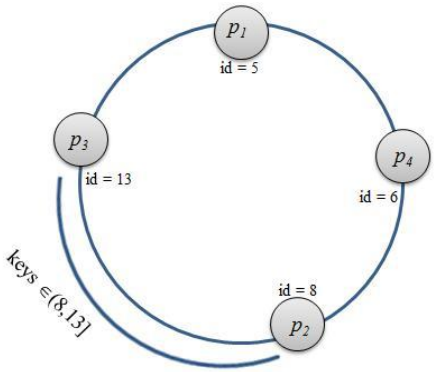


Figure 2.6. Example of DHT

2.3.3 Dynamic P2P Networks

Dynamic P2P networks do not have stable links among neighbors, and the contacts of a peer are changed continuously. The goal is to enable each user to construct

and update dynamically a potential view of the overlay. Gossip protocols are used to manage and construct the overlay [78].

Gossip protocols, also known as epidemic protocols, have been initially proposed to maintain the mutual consistency of replicated data by spreading replica updates to all nodes over the network [5]. Basic gossiping is simple. Each node in the network has a complete view of the network (i.e., a list of all nodes' addresses) and chooses a node at random to spread the request. The main advantage of gossiping is robustness over node failures since, with very high probability, the request is eventually propagated to all the nodes in the network. In large P2P networks, however, the basic gossiping model does not scale as maintaining the complete view of the network at each node would generate very heavy communication traffic. A solution to scalable gossiping is to maintain at each node only a partial view of the network, e.g., a list of tens of neighbor nodes. To gossip a request, a node chooses, at random, a node in its partial view and sends it the request. In addition, the nodes involved in a gossip exchange their partial views to reflect network changes in their own views. Thus, by continuously refreshing their partial views, nodes can self-organize into randomized overlays that scale up very well.

Gossip protocols have been successfully used in P2P networks to solve a wide range of issues such as P2P overlay construction and maintenance [50][145] [68], data dissemination [76][77][45], data aggregation [74][67][89], data replication [63], and resource monitoring [146][73]. They have shown high interest in designing new P2P networks because of their scalability, robustness, simplicity, load balancing, and resilience to failures [78].

In a system where gossip protocols are used to construct and maintain the P2P overlay, each peer maintains a set of entries in its view. Each entry refers to a peer and includes information about the data shared by that peer (e.g. shared documents, contacts, etc.) called view, with another randomly selected peer. Usually, the number of entries in a peer's view is very small and less than the network size. With gossip, each peer periodically exchanges a subset of its view, with another peer, and updates its view accordingly. In this thesis, we exploit gossip protocols to construct, maintain overlay and disseminate information, to derive scalable, efficient, high quality, and reliable P2P recommendation systems.

The behavior of a gossip protocol running at each peer can be modelled with two separate threads: *active* and *passive behavior*. The active behavior describes how a peer p_i initiates a periodic gossip exchange message, while the passive behavior shows how the peer p_i reacts to a gossip exchange initiated by some other peer p_j. Typically, in gossip protocols, each peer p keeps locally a view of its dynamic acquaintances (or view entries), and their corresponding shared data, and p uses gossiping to exchange and update its view. Generally, gossip protocols consist of three modules: *selectContact, exchangeInfo,* and *updateInfo*.

- **selectContact** defines the way a peer p_i selects another peer p_j to gossip with. Peer p_j can be selected randomly or based on a biased criterion that is depending on the application. This module is performed by the peer that runs the active behavior.
- **exchangeInfo** defines the way the information is exchanged between the gossiping peers. The information exchange can be performed with two strategies: push

and pull. With push, the peer that runs the active behavior shares its information with the remote peer. With pull, the peer that runs the passive behavior shares its information with the gossip initiator. Gossip protocols can perform either strategy or a combination of both.

- **updateInfo** defines how peers use the information received via gossiping to update their views. It can be performed by the peer running either active or passive behavior.

Figure 2.7 illustrates a pull-push gossip exchange between peers p_1 and p_2. The figure shows that each peer p_i maintains locally a view, which includes a set of contacts along with their shared data. For instance, peer p_1 maintains locally a view, which includes information about peers p_3, p_2, and p_4. Peer p_1 initiates a gossiping exchange as follows: *p1* selects peer p_2 to gossip with. Then p_1 selects a subset of its view and sends it to p_2, which in turn does the same. Finally, when p_1 receives the gossip information, it merges it with its view, and updates its view.

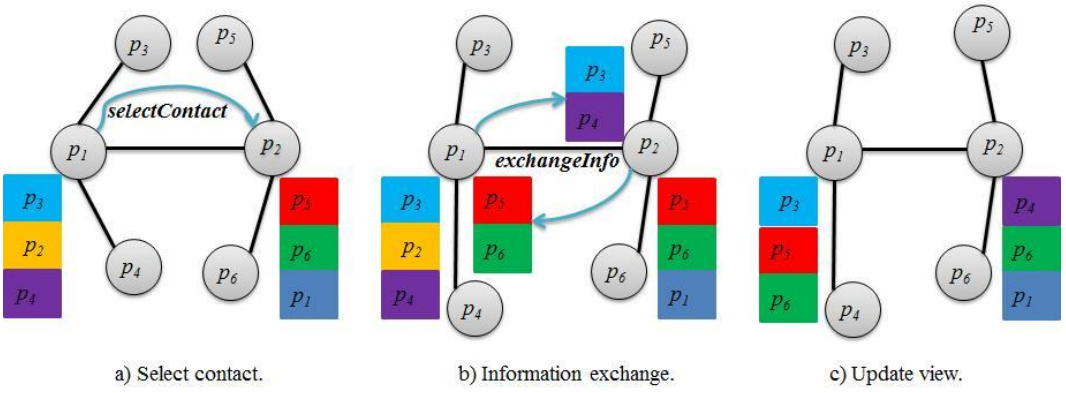


Figure 2.7. Pull-push gossip exchange between peers p_1 and p_2

2.3.4 Requirements for P2P Recommendation Systems

Recommendation systems have been widely used to filter out the items the user may like from a huge number of items based on user's interests and preferences. P2P online communities have become popular in sharing contents, and this popularity has translated into large amounts of data being spread over high numbers of peers (and users). Using a centralized RS may suffer the traditional problems of centralized infrastructures: single point of access, which may hurt performance and single point of failure, which may hurt availability.

Our objective is to provide a decentralized RS for large-scale data sharing for P2P online communities, where each peer represents a user labelled with the contents, ratings, social data, etc. it stores. Each user is responsible for storing locally its contents, ratings, social data, etc. and collaborates with other users in order to facilitate

searching, generation of recommendations, and self-organization without central control.

The main requirements to be fulfilled to design a P2P-RS for online communities are *reliability, scalability, performance, expressiveness*, and *quality* of recommendations

- **Reliability** guarantees that each user can receive recommendations, and each item has the possibility to be recommended. To support reliable recommendations, the system should have the ability to cope with the sparsity and cold start problems. Thus, users' contents and social data should be exploited in a way that insures that each user has a set of neighbors even though the user has not rated items yet, and each item may have similar items even though the item has not been rated yet.
- **Scalability** refers to the ability of the system to accommodate the increase in the numbers of users and data. Therefore, the system should operate normally (and prevent bottlenecks) when the numbers of items and users increase. The choice of the P2P overlay has direct impact on the scalability of the system. The system should self-organize the users and data in the overlay, in a simple way that incurs little overhead and network traffic.
- **Quality** of recommendations guarantees that each user receives high quality and valuable recommendations that are related to its interests and requests. Quality is affected by the users' preferences and social data, and content semantics that are used to construct neighbors. Thus, the system should have the ability to exploit content semantics, users' preferences such as ratings, feedbacks, and users' social data such as friends, trusts etc. Moreover, the system should give each user the ability to rate the content it explores, to return feedbacks on the recommendations it receives, and to establish friendship with other interested users with a declared trust.
- **Performance** guarantees that each user receives recommendations as quickly as possible, and reduces the overhead and resource consumption. The performance of the system is mainly affected by the way the data are aggregated; the choice of the P2P overlay, and the way used to route users' requests in the overlay. The system should disseminate users' data, aggregate users' neighbors, and organize users in the overlay in a way that can reduce the overhead, resource consumption, and search time. For that, the system should aggregate for each user *u* the users that may help *u* in getting recommendations, and only search the users who have items that are related to the user requests.
- **Expressiveness** guarantees that each user has the ability to interact with the system, and express its demands, and thus avoiding users from receiving the same list of recommendations many times. The system should give users ability to determine its demands through keywords query. Moreover, the system should have ability to express the semantics of the query and contents.

2.4 P2P Content Management Systems

Several approaches have been proposed to build distributed information retrieval systems using P2P networks. These systems, called *P2P content management systems*, express query by a set of keywords, and use the semantic of contents to build the overlay and locate peer and contents place.

P2P systems such as Gnutella, and KaZaA (e.g. file sharing), support limited functions (e.g. a very simple keyword search capability, trying to find the contents whose name or description match the keywords provided by the user) and use simple routing techniques (e.g. resource location by flooding) that have performance problems. To improve the query expressiveness and performance of search**,** guided search approaches have been proposed [4][31]. Therefore, P2P content management use information retrieval techniques to represent, store, organize and recover specific information from stored data. These systems are classified according to the construction of their overlays in two categories.

- **Clustering overlays**: exploit the contents stored at user peers to group similar peers or similar data in one logical cluster.
- **Shortcut link overlays:** whereby peers establish direct links with other peers that are similar with respect to interest or social patterns. These links can either replace or be added on top of a peer neighborhood.

  In the rest of this section, we describe in more details these two kinds of overlays.

2.4.1 Clustering Overlays

In these systems, content semantics is exploited as clustering criteria. These systems use either *peer clustering* or *data clustering*. In peers clustering, peers that are similar (in terms of content) are grouped together while in data clustering, contents that have some similarity are placed at the same peer.

Peer Clustering: In systems like [30][16][64][83][70][150], peers are organized in similar content clusters on top of an unstructured overlay (i.e., peers with similar contents are grouped in one cluster). A query is guided to a cluster that is more likely to have answers to the given query, and then the query is flooded within this cluster. For instance, Garcia-Molina et al. [30] introduce the concept of semantic overlay by using clustering and classification techniques, in order to improve the performance of search. They explicitly identify a predefined classification hierarchy to classify the peers' documents. Each peer joins the system, identifies which cluster(s) to join by acquiring the classification hierarchy in the system using flooding. Then, it classifies its documents against this classification hierarchy. Finally, the peer uses flooding to find peers that belong to its cluster, and then it joins its clusters.

The SETS System [16] clusters peers based on their documents' topics. A fixed set of C topic segments is predetermined, where each topic segment represents a cluster, and all peers in a cluster $c \in C$ store documents related to the topic segment of cluster c. SETS uses a vector space model to represent documents, peers and cen-

troids. Centroids are used to represent the topic of a cluster *c* and a peer *p*. The centroid of a cluster *c* is computed by averaging the vector terms of the documents that belong to cluster *c*. Similarly, the centroid of a peer *p* is computed by averaging the vectors terms of *p*'s documents. The knowledge of the *C* centroids is global, and given to a distinguish peer. Once a peer *p* joins the system, it identifies its cluster by measuring the distance between its centroid and the *C* centroids, and then joins the most similar clusters. When a peer *p* issues a query *q*, it forwards *q* to all members of its clusters using flooding.

Data Clustering: In systems like [28][94][97][129][101], similar contents is placed with respect to their semantic on the same node. Typically, these systems use a structured overlay to place similar content in the same area. PSearch [28] uses latent semantic indexing [32] to distribute documents in the CAN DHT. Latent semantic indexing is an indexing and retrieval method used to cluster a set of documents into a number of groups by analyzing the relations between documents and the terms they contain using a mathematical technique called singular value decomposition.

Hence the documents that are semantically similar also appear close to each other in the CAN system. Accordingly, documents that are relevant to a query are likely to be collocated on a small number of nodes. Sahin et al. [129] represent each document as a vector of terms and uses them to determine the location of the document indices in a Chord DHT.

2.4.2 Shortcut Link Overlays

In these systems, each peer establishes logical links (i.e. shortcut links) to peers that may have contents related to its queries on top of its neighborhood. A query is forwarded first to those links. If the user is not satisfied or the search fails, then the query is routed through the traditional flooding techniques. We classify these systems into *interest-based* and *social-based* based on the data that is used to create the shortcut links. In interest-based, shortcut links are created based on query histories or peer interests. Peer interests are defined either explicitly or implicitly. Social-based systems use users' social data such as common behaviour patterns, tags, bookmarks, friends, etc. to create the shortcut links.

Interest-based: These systems rely on the assumption that if a peer p_i has content that another peer p_j requested or is interested in, then most probably p_i has other contents that p_j is interested in. Thus, in these systems each peer *p* adds links to peers that interests are similar to *p*'s interests, or that have successfully answered *p*'s queries [147][158][161][65][167]. Upadrashta et al. [158] propose to let each peer add shortcut links to similar peers in terms of interests, in top of Gnutella, to enhance its search. In this model, each peer is interested in a set of categories of interest. A category of interest is defined as a bag that consists of keywords or topics. Peer's categories of interests are either extracted implicitly from peer's documents, or stated explicitly by the user. When a peer *p* submits a query *q* in a category of interest *t*, it creates a shortcut link to peers that successfully answered *q*. Each peer maintains *n* peers that have strong relationships for each category of interest, and uses it for searching files

in the network. According to their results, the shortcut links reduce search time for queries as well as the number of messages circulating in the system.

Similarly, in [147] and [65], each peer *p* adds shortcut links, on top of Gnutella, to the most recent peers that have successfully answered *p*'s queries. When *p* searches for contents, it disseminates its query to its shortcuts and, if the search fails, uses flooding to search the underlying P2P overlay.

Social-Based: These systems consider user social behaviours and data for constructing the overlay and searching contents [13][47][99][157][115][22][121]. Fast et al. [47] propose a P2P overlay for music file sharing that is constructed based on users' preferences on music style. A hierarchical Dirichlet process [151] is used locally to extract the music styles of each user *u* from the music files it stores. The authors start from a random overlay network, where each user is connected to a set of neighbors. Then, when users start searching for music files, shortcut links are established between users that have similar music styles. When a user *u* initiates a query *q*, *u* forwards *q* to each neighbor *v*, which in turn forwards *q* to its neighbors until TTL. Then, *u* establishes a shortcut link to each user *v* that has responded to *q* successfully, and *v*'s music styles are similar to *u*'s music styles. Once *u* has established shortcut links, it uses them to serve its queries.

SPROUT [99] exploits users' explicit friends on top of the Chord DHT, in order to avoid misrouting, and increases the number of query results. When a peer *p* joins the DHT, in addition to its routing table, *p* adds shortcut links to all its online friends. Once *p* issues a query *q* with key *k*, it forwards *q* to the friend whose id is closest to, but not greater than, *k*. In turns that friend forwards *q* in the same manner, until the peer which is responsible for *k* is found. In case *p* does not find a friend that is close to, and less than *k*, it uses the regular Chord algorithm.

In [115], the authors propose to use personalized PageRank [114] in order to give users the ability to perform personalized search in an unstructured P2P network. Each peer computes the PageRank scores of its pages in a distributed manner, and the peer is aware of the rank scores of the pages owned by its neighbors. Once a peer p_i issues a query *q*, it forwards *q* to the neighbor that has the highest ranked page. Each peer p_j that receives *q* forwards *q* to the neighbor which has the highest ranked page. This process continues until the desired number of results is obtained.

Bai et al. [13] propose a personalized P2P top-k search for collaborative tagging systems, called P4Q. In P4Q, each user *u* maintains locally a profile, denoted by *profile(u)*, which includes the items that *u* has tagged along with their tags. In addition, it maintains a *personal network*, denoted by *network(u)*, that includes a fixed number *n* of users with similar interest. The *network(u)* consists of two parts (see Figure 2.8), the first includes the *c* users that have the highest similarity with *u*, denoted by *profileList(u)*, along with their profiles such that $c << n$. The second part includes the other similar users along with the bloom filters of their profiles, denoted by *bloomfilterList(u)*. Two users are considered similar if they share a common number of tagged items.

In order to find and construct the *network(u)* of user *u*, two gossip protocols are used. The first is used as peer sampling to keep the overlay connected. The second is used to gossip the bloom filter of users' profiles, and measures the similarity between them based on the bloom filters of their profiles. Once *u* has determined the *c* users with highest similarity, it stores them locally along with their profiles in its *pro-*

fileList(u), and then stores the other similar users along with the bloom filters of their profiles in its *bloomfilterList(u)*.

When a user *u* issues a query (which is a set of tags), it uses the profiles of the *c* users in its *profileList(u)* to process locally its query. If the search fails, or *u* is not satisfied with the results, *u* gossips its query as follows. First, *u* selects from its *bloomfilterList(u)* the users that have tags similar to the query's tags, and adds them to a list, called *remainingList(u)*.

In the example of Figure 2.8, user *u* selects the users *v*, v_4, v_6, and v_8 from its *bloomfilterList(u)*, and adds them to its *remainingList(u)*. Once *u* has selected the members of its *remainingList(u)*, it selects randomly a user *v* from the *remainingList(u)*, and forwards the query along with the *remainingList(u)* to *v*.

When a user *v* receives a query message issued by a user *u*, it checks whether it stores locally a user *x* in its *profileList(v)* given that $x \in remainingList(u)$. If *v* finds the user $x \in remainingList(u)$ in its *profileList(v)*, it sends *x*'s profile to *u*. Next, *v* removes *x* from the *remainingList(u)* (see Figure 2.8). Then, if the *remainingList(u)* is not equal to zero yet, *v* splits the *remainingList(u)* into two parts after removing itself from the *remainingList(u)*, and returns the first part to the query initiator *u*. Finally, *v* selects randomly a user *y* from the second part of the *remainingList(u)* and forwards it along with the query to *y*. Notice that, the query initiator *u* keeps gossiping its query while its *remainingList(u)* is not equal to zero.

In Figure 2.8, user *v* splits the *remainingList(u)* it received into two parts: the first part includes user v_6, and the second part includes user v_8. After that, *v* returns the part that includes v_8 to the initiator *u*, and forwards query to the user v_6.

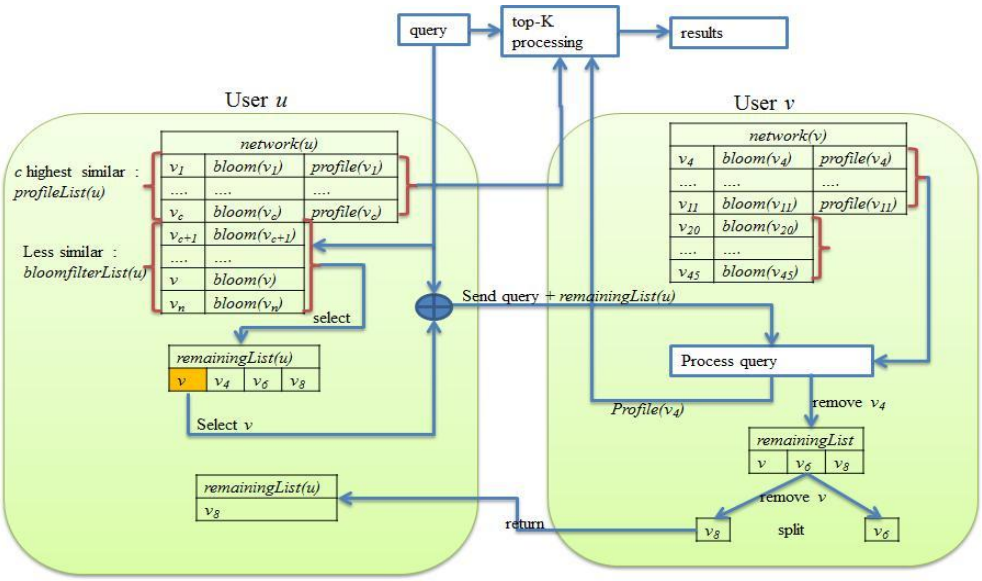


Figure 2.8. Query processing in P4Q

2.5 P2P Prediction Systems

In centralized RSs, the time complexity in finding the closest neighbors, and the space required to store user-item matrix increases exponentially when users and items increase. Decentralized architectures such as P2P systems that provide scalability, and fault tolerance for large-scale applications, have thus been exploited to build distributed RSs.

In P2P prediction systems, the user-item matrix, users' profiles and social data are distributed over the participant users, and each user stores its own data and a part of other users' data. Then users collaborate to share the data they store and to generate recommendations. We classify P2P prediction systems according to the data that are used to construct the P2P overlays and generate recommendations, into the following category:

- **Basic P2P prediction systems**: In these systems, recommendations are generated based on users' ratings only.
- **Social P2P prediction systems:** These systems consider users' social behaviors in constructing the overlay and generating recommendations.

In the rest of this section, we describe in more details these two kinds of prediction systems, and show their limitations.

2.5.1 Basic P2P Prediction Systems

Recall that RSs that use a centralized infrastructure use a single service provider to store the user-item matrix (see Section 2.2.2). Also that server or operator is responsible for similarity computation and recommendations generation. In basic P2P prediction systems [106], each peer keeps a fraction of the user-item matrix. When a peer needs recommendations, first it aggregates the other fractions of the user-item matrix from the other peers. Then, it performs locally the similarity computation to extract its neighbors. Finally, it generates recommendations with the help of the neighbors it has extracted.

The first work on distributed RSs is Tveit [156], a P2P collaborative-based filtering system to suggest recommendations for mobile customers. The system is based on a pure P2P topology like Gnutella, and queries that include users' ratings are simply propagated by flooding. In addition, each user u maintains locally a cache that includes a copy of each query q that u has received, i.e., when the user u receives a query q that includes a user v's rating data, u maintains locally q in its local cache. In Tweit, a user u generates recommendations as follows. First, u sends a query q that includes its rating data to each neighbor. Each user v receives the query q, it measures the similarity between q and each query q^* that v has stored in its local cache using the cosine similarity, and returns to u each q^* whose similarity with q exceeds a specific threshold. Then, v stores q in its local cache, and forwards q to each of its neighbors if TTL is not equal to zero yet. Once u has received the ratings data that are simi-

lar to its ratings data, it generates recommendations by computing the weighted sum of those ratings data, and makes suggestions based on those ratings data.

In PocketLens [106], a P2P-RS, the authors propose four architectures including random discovery (similar to Gnutella), transitive traversal, distributed hash table (DHT), and secure blackboard. In all these architectures, a user *u* searches the P2P overlay to find new neighbors. Then it aggregates the ratings data from those neighbors. After that, *u* measures the similarity between itself and those neighbors, and selects the top-k similar users for generating recommendations. The Pearson correlation coefficient is used to measure the similarity between users based on their ratings data. User *u* generates recommendation as follows. First *u* selects the items that have been rated by the top-k similar user, and not seen by *u* yet. Then, *u* measures the similarity between those items and its items, and selects the top-n similar items as the recommendation results.

The architectures proposed in PocketLens differ in the way they search for new neighbors. In random discovery, when a user *u* joins the system, it sends a *ping* message to each neighbor. Each neighbor *v* that receives a ping message returns to *u* a *pong* message that includes all its neighbors. When *u* receives a pong message, it aggregates the rating data from the users in that pong message. In the example of Figure 2.9, *u* forwards a ping message to its neighbors *v*. User *v* returns to *u* a pong message that includes *v*'s neighbors. Thus, *u* becomes knowing the existence of users v_1 and v_2 after the one round of pingpong messages.

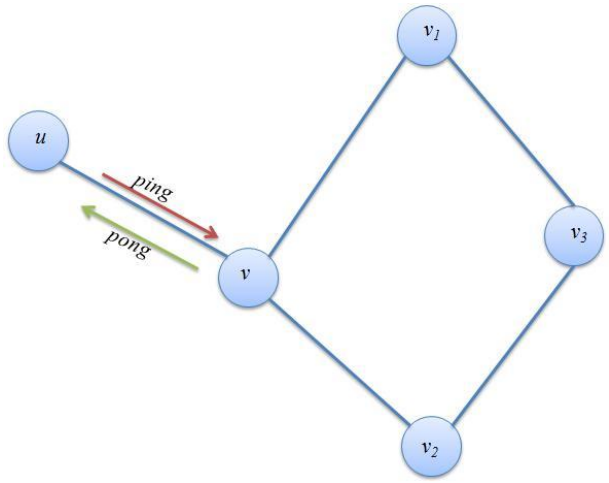


Figure 2.9. Ping/Pong in the random discovery architecture of PocketLen

Transitive traversal lets each user *u* find new neighbors during the time of searching for contents. In this architecture, *u* uses the flooding from Gnutella to forward its query in the overlay, and *u* considers each user *v* in the response path of the query as a a new neighbor. In content addressable architecture, distributed hash table (DHT) is used to publish users' rating data. The items along with their ratings and similar items are stored at DHT nodes using the items identifier to identify the keys. When a user *u* generates recommendations, first it aggregates the items it has rated along with their ratings and similar items from the DHT. Then it generates the recommendation re-

sults. The last architecture (secure blackboard) lets each user encrypt each item it shares, in order to increase user's privacy. Simulation results show that PocketLens produces good recommendations, and the results are close to the results obtained by using a centralized recommendation system.

Peng et al. [118] propose to store users' ratings over a DHT in order to distribute the user-item matrix, called *PipeCF*. All users that have rated an item with the same rating are grouped in one cluster, called *bucket*. These buckets are spread over the DHT. The item's name and rating are used to identify the *key* of each *bucket* (see Figure 2.10). A user aggregates all buckets of its items to make recommendations.

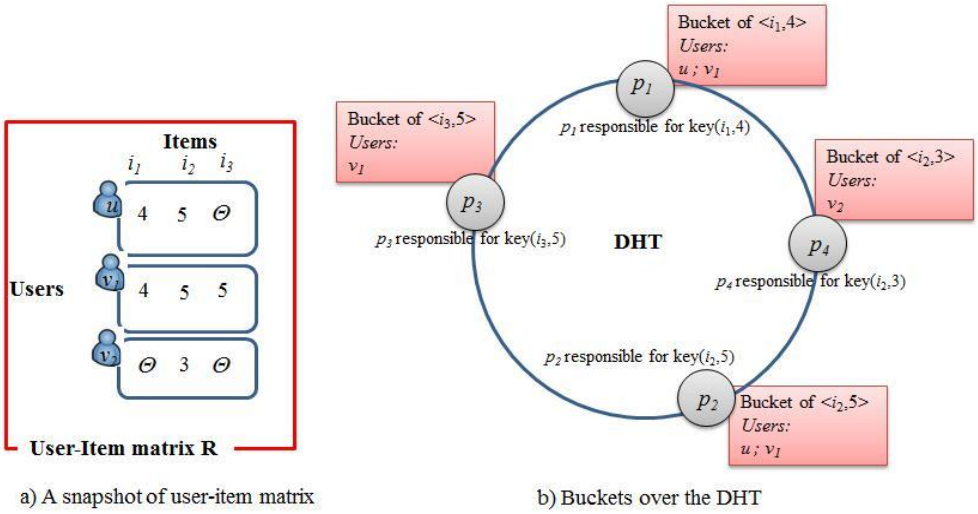


Figure 2.10. A snapshot of the PipeCF system

Kermarrec et al. [79] use gossiping and random walk for a decentralized RS. Gossiping lets each user *u* aggregate the most similar users, denoted by *neighbors(u)*. Information about them is stored locally along with their profiles. A user's profile includes the items the user has rated along with their ratings. Cosine similarity is used to measure the similarity between users.

Each user *u* computes its recommendations by running locally a random walk on its neighbors and neighbors of neighbors as shown in Algorithm 2.1. User *u* inputs to the algorithm the random walk probability α, to decide whether to continue random walking or stop. The transition probability matrix *P* is defined over *neighbor(u)*. Each entry P_{vx} represents the probability that user *v* would ask user *x* for recommendations and is defined as:

$$P_{vx} = (1-\beta)\frac{sim(v,x)}{\sum_{z \in neighbor(u)} sim(v,z)} + \frac{\beta}{m} \qquad (2.7)$$

where β is a scale parameter such that $\beta \in (0,1)$, to give the user ability to jump randomly to another user in its neighborhood during the random walk.

In addition, *u* inputs to the algorithm the initial vector d_u of the probability distribution over *neighbor(u)*. Each component of d_u is defined as:

$$d_u(v) = \frac{sim(u,v)}{\sum_{z \in neighbor(u)} sim(u,z)} \quad (2.8)$$

The output of the algorithm is the final vector d_u^* of the probability distribution over *neighbor(u)*, and is defined as:

$$d_u^* = d_u \alpha p (1 - \alpha P)^{-1} \quad (2.9)$$

where $(1-\alpha P)^{-1}$ is the inverse of the matrix.

The entries of the final vector d_u^* is used as the similarity weight between *u* and its neighbors, in order to predict the rating of unseen items and generate recommendations. The simulation results using the MovieLens dataset show that random walk enhances the precision of recommendation results. However, computing the transition similarity matrix of the random walk and its inverse is complex, and time consuming, as the number of items and its neighbors (and neighbors of neighbors) increase.

Algorithm 2.1- Random walk at user *u*

Input: random walk probability α; the transition probability matrix *P*; initial vector d_u of the probability distribution
Output: the final vector d_u^* of the probability distribution.

1 **Initialize** $x = d_u$
2 **While** *x* has not converged
3 $x = d_u \alpha p (1 - \alpha P)^{-1}$
4 **End while**
5 $d_u^* = x$

2.5.2 Social P2P Prediction Systems

These systems combine users' preferences (ratings) with users' social data (friends, trust, etc.) in order to improve recommendations performance. Kim et al. [80] propose a P2P-RS based on FOAF files. This system is designed to generate movie recommendations. Each user *u* extracts locally its preferences, and asserts them in its FOAF file. User *u*'s preferences consist from the movie genres *u* has rated. Where movie genres are divided into 19 categories, and *u* computes its preferences in each genre based on the ratings that *u* has given to the movies in that genre. The system uses users' FOAF files to cluster the users with similar preferences in one group on real-time, and recommendations are propagated automatically from user to user, starting from an initiator. When a user *u* rates an item *i*, it recommends the item *i* to its friend as follows. First, the system groups the friends that are similar to *u* by aggregating the FOAF file of each friend *v* of *u*. Then, the similarity between *u* and friend *v* is computed by using the cosine similarity between *u* and *v* preferences. Then, the group is established, in which it includes each friend *v* that its similarity with *u* exceeds a system-defined threshold. Finally, *u* forwards its recommendation to each friend *v* in that group.

Once a user *v* receives a recommendation for item *i* from *u*, it either rates or does not rate *i*. In case *v* has rated *i* with a rating that exceeds a specific threshold, it recommends *i* to its friends as described before. Otherwise, *v* stops recommending *i*. The recommendations are propagated until a specific depth *n* is reached, where depth *n* is the number of hops between the recommendation's initiator and terminator.

Massa et al. [100] propose a trust-aware collaborative-based filtering system that uses users' trust data in a decentralized environment such as P2P online communities (e.g., Slashdot.org [143]) in marketplace sites (e.g., eBay.com [98]) in order to enhance the performance of recommendations. The system lets each user *u* express its level of trust, denoted by *trust(u,v)*, with each user *v* it has interacted with. The *trust(u,v)* value is between 0 and 1, where 0 means total distrust and 1 means full trust. The trust network is represented as a directed graph *G=(U,E)*, where *U* is the set of users in the network and *E* is the set of edges between users. There is an edge *e(u,v)* from user *u* to user *v*, if *u* has expressed its level of trust on user *v* (see Figure 2.11). The trust level from user *u* to user *v*, for which there is no direct edge trust, is predicted based on a maximum propagation distance *d* (system defined), and the minimum distance *n* between *u* and *v*, and is defined as:

$$trust(u,v) = \frac{d-n+1}{d} \tag{2.10}$$

where distance is computed based on the number of hops.

In the example of Figure 2.11, we assume that the maximum propagation distance *d* is 4. Then, the predicted trust value from *u* to *v* is 0.75, and the predicted trust value from v_2 to *u* is 1. When *u* measures the trust level between itself and each user in the network, it selects the most trustful as its trustful neighbors, denoted by $neighbor_{trust}(u)$. Those neighbors are used to compute the recommendations. When *u* predicts the rating for an unseen item *i*, it selects those users that have rated item *i* from $neighbor_{trust}(u)$. Then, *u* computes the weighted sum of those users' ratings, and makes suggestion based on those ratings. Simulation results using Epinions.com data set show that trust aware recommendation alleviates the cold star problem, and increases the number of returned recommendations.

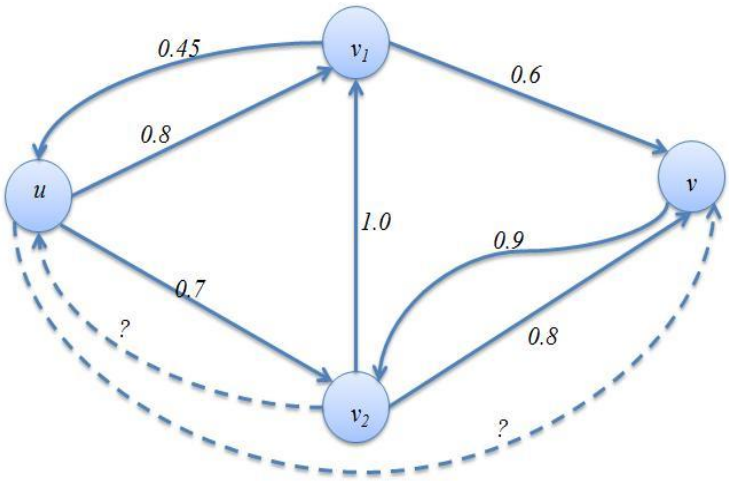


Figure 2.11. A snapshot of a trust network

Sarda et al. [133] propose a distributed trust-based recommendation system for social networks. The system recommends a vendor for an item the user likes to purchase. Each user *u* maintains locally a list of friends, and the system allows *u* to express its level of trust to each friend *v* it has maintained, denoted by *trust(u,v)*. The value of a trust given by a user *u* over a friend *v* is varied between 0 and 1. In addition, each user *u* maintains locally the values of trust between itself and each user *v* in the system. Given that these trust values are computed beforehand by using TidalTrust algorithm [52] i.e., no transitive trusts are computed at run-time.

TidalTrust uses a modified breadth first search-based algorithm to infer the trust value from *u* to *v*, if *u* has not expressed its trust level with *v* yet. TidalTrust finds all possible paths between *u* and *v*, and computes the trust value along each path by multiplying the trust values between direct users along the path. Then it computes the trust value from *u* to *v* by averaging the trust values along all paths.

The system allows each user *u* to give a rating value between 0 and 5 for each item *i* that it has explored, denoted by r_u^i. When an originator user *u* initiates a query *q* seeking for recommendation for an item *j*, it forwards *q* to each neighbor *v* such that the trust value *trust(u,v)* exceeds specific threshold.

Each user *x* receives *q* from a neighbor *v*, processes *q* as follows. First, *x* checks if it has rated the item *j*. If this is the case and r_x^j exceeds a specific threshold, *x* sends back r_x^j to *v*, and stops forwarding *q*. Otherwise, *x* forwards *q* to each neighbor *y* such that *trust(x,y)* exceeds a specific threshold.

When user *v* receives a response from a neighbor *x*, it computes a score for that response as the product between *trust(v,x)* and r_x^j. Responses trace back along the path of *q* until the query's originator *u*. Finally, the query's originator *u* selects the response that has the highest score and its corresponding user as a result for the query.

Wang et al. [163] propose a P2P-RS for television systems on top of Tribler [121]. Each user *u* maintains locally a set of top most similar users, called *buddy(u)*, and a set of random peers along with their profiles. Whenever a user selects another user to contact, it first merges its *buddy* with the random peers 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 users to be selected and gives a chance for new users to be explored.

User profiles consist of the users' interests that are extracted from the watched TV programs. User *u* is interested in a TV program, if the time that *u* spent in watching that program divides the duration time of the program, and the number of times that program has been broadcast exceeds a specific threshold. A user *u*'s profile, denoted by *profile(u)*, contains a set of *K* TV programs, where *K* is the number of unique TV programs, and each TV program $k \in K$ is associated with a Boolean value that indicates whether *u* is interested in that TV program.

Once the user *u* has aggregated its *buddy*, it exploits them to generate recommendations. Algorithm 2.2 shows how a user *u* generates recommendations taking into account its profile and buddy. User *u* selects each TV program *k*, such that $k \notin K_u$, where K_u is the set of TV programs that *u* has watched, and then computes the rank of *k*, denoted by *rank(k)*, as follows. First, *u* computes the popularity of *k*, by counting

how many users in its buddy is interested in *k*. Then, *u* counts how many users in its buddy are interested in *k* and in each TV program k^* such that $k^* \in K_u$, and then sums it to the popularity of *k*. Finally, *u* orders the TV programs that *u* has not watched yet in a list, and selects the top-n TV programs as the recommendation result.

Algorithm 2.2- Generate recommendation at user *u*

Input: *profile(u); buddy(u)*
Output: recommendation result

1. **For** each $k \notin K_u$ **do**
2. **For** each user $v \in buddy(u)$ **do**
3. **If** *v* is interested in *k* **then**
4. **Increment** popularity of *k*
5. **End if**
6. **End for**
7. **For** each $k^* \in K_u$ **do**
8. **For** each user $v \in buddy(u)$ **do**
9. **If** *v* is interested in *k* and k^*
10. **Increment** *k's* co-occurrence
11. **End if**
12. **End for**
13. **End for**
14. *Rank(k)* = popularity of *k* + *k*'s co-occurrence
15. **Add** *rank(k)* to a **List**
16. **End for**
17. **Order** the *List*
18. *Recommendation* = the top-n of the ordered *List*

2.6 Summary and Observations

In this section, we summarize our review of the work related to this thesis and our observations, which will be useful to introduce our approach.

The solutions we discussed, shown in Figure 2.12, fall either within decentralized or centralized infrastructures. In centralized infrastructures, the users and their rated items are modelled in a user-item matrix, and stored at a single service provider, which also performs all the recommendation processes. On the other hand, decentralized infrastructures distribute the user-item matrix over users, and users cooperate to generate recommendations.

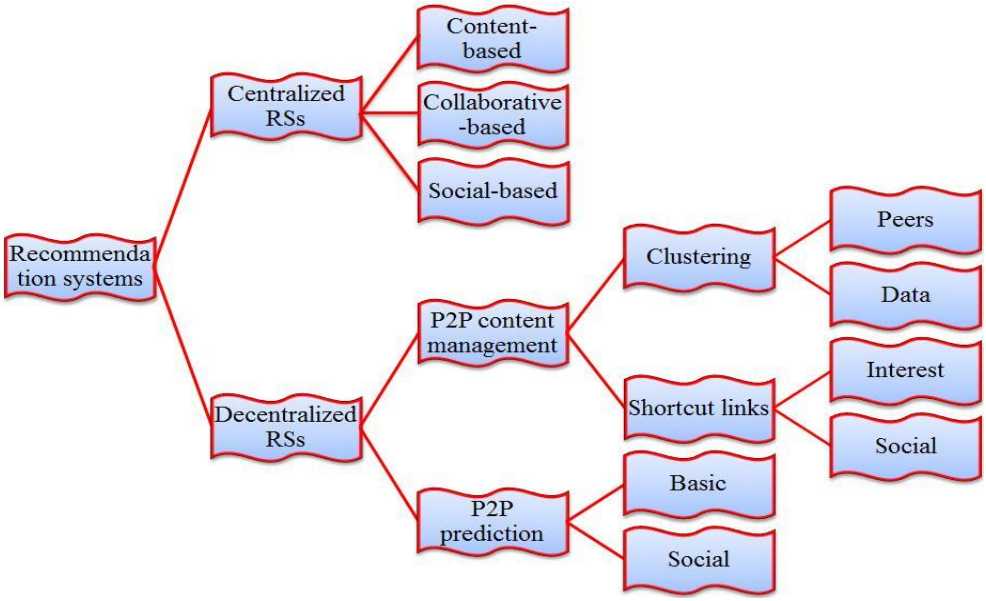


Figure 2.12. Recommendation systems and a hierarchy of solutions

We observed that the three classical centralized RSs, namely *collaborative-based filtering*, *content-based filtering*, and *social-based filtering*, consume large storage space, suffer from limited scalability, and require high cost in maintaining and updating the underlying infrastructure. Moreover, centralized *collaborative-based filtering* and *content-based filtering* consume much time in measuring similarity and suffer from sparsity and cold star problems.

Decentralized infrastructures encompass two trends, namely, *P2P content management systems* and *P2P prediction systems*. We observed that neither P2P content management systems nor P2P prediction systems fully satisfy the requirements which we identified in Section 2.2. In P2P content management systems, users send keyword queries to the system that returns a set of contents that are most related to query. The solutions in these systems focus on reducing the time and network traffic consumed by search, but they do not take into account users' feedbacks and ratings, which may deteriorate the quality of results.

P2P prediction systems proactively return a set of recommendations to users based on their profiles. The solutions in these systems focus on how to distribute the user-item matrix over users, but they do not take into account the overhead required to aggregate the user-item matrix by users, which also may deteriorate the performance and scalability of the system. In addition, they do not give users the ability to express their demands, thus hurting expressiveness.

Figure 2.13 illustrates the services provided by a P2P infrastructure, and the modules required for each category of solutions previously described. The services provided by the P2P infrastructure are *overlay*, *location* and *routing*. The P2P *overlay* defines the neighbors of each user and how users are connected. The *location* service defines the location of users and contents in the overlay. And the *routing* service defines how users forward their discovery messages and queries.

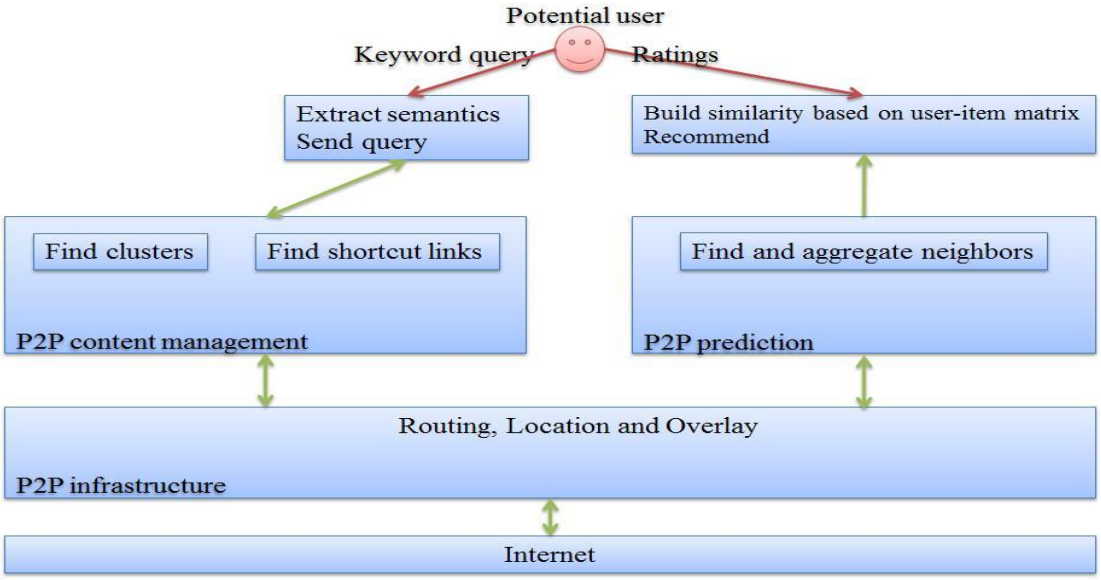


Figure 2.13. Overview of the P2P recommendation systems architecture

P2P content management systems use information retrieval techniques such as clustering to enhance the search for contents in P2P networks. Two approaches are identified: clustering overlays and shortcut link overlays. In clustering overlays, peers that are similar with respect to contents are grouped in one cluster usually on top of an unstructured overlay, or similar contents are placed on the same peer usually on top of a structured overlay. The clustering module searches the P2P overlay to find the location (cluster) of a peer or content. In unstructured overlays, searching for the clusters of users is achieved through flooding, which increases network traffic overhead. In structured overlays, the content semantics are used so as to determine the peer responsible for the location of that content. Due to the location constraint, data clustering requires high maintenance overhead, and gets less efficient as the size of the shared contents increases. However, clustering overlays enhance performance by reducing the number of messages in the network for a query, and provide rich query expressiveness.

Shortcut link overlays let each user establish logical links to friends or to users with similar interests, usually on top of an unstructured or dynamic overlay. The shortcut link modules search for potential users in order to establish links with them. These systems use users' interests or social data in order to establish new shortcut links. We observed that using users' social data such as friends, trust, etc. increases the quality of results, and enhance the performance of the system. In unstructured overlay, flooding or query propagation is used to search for potential users, and does not have any mechanism to manage the number of shortcut links, in which it may deteriorate the scalability of the system. In dynamic overlay, gossip protocols are used to construct users' shortcuts. We observe that using gossip protocols give users ability to self-organize themselves, and increase the scalability of the system. However, exchanging users' profiles, and storing them locally may increases the traffic consumed in the network and storage space.

Along these lines, integrating users' social data or using gossip protocols in P2P content management systems may increase the scalability, quality and performance of the system.

In P2P prediction systems, users' ratings and social data are distributed over peers, and are used to compute the recommendations. Two approaches are identified in this category: basic and social P2P prediction systems. Basic P2P predictions systems just distribute the user-item matrix over the peers, and then peers collaborate and communicate in order to compute the recommendations. Social P2P prediction systems leverage users' social data such as trusts, friends, etc. with the user-item matrix in order to compute recommendations.

The "find and aggregate neighbors" step searches the P2P overlay to find neighbors, and then aggregates their profiles in order to let each user locally build the user-item matrix. Most of the basic and social P2P prediction systems are built on top of an unstructured overlay, and flooding is used to find neighbors. We observed that aggregating neighbors' profiles (users' preferences, FOAF files, etc.) and using flooding to find neighbors increase the overhead traffic, which may hurt the scalability of the system. Structured overlays reduce the network traffic for aggregating neighbors' profiles, but increase the maintenance overhead because of their tightly controlled topology.

We observed that basic P2P prediction systems have limited reliability due to the sparsity and cold star problems, and limited quality due to the lack of social relations such as friends, trusts, etc. between users. Social P2P prediction systems have increased the reliability and quality of the results. However, both basic and social P2P prediction systems have limited performance due to the traffic and time required to find and aggregate neighbors' profiles, and suffer from expressiveness because they do not give users ability to query the system. Along these lines, integrating users' social data or users' ratings, feedbacks, rankings, etc. increases the quality and the reliability of the returned results.

Finally, we observed that taking into account contents' semantics alleviates blind search because only the peers that can serve queries are exploited, increases query expressiveness, and increases the reliability of the results by reducing the effects of the sparsity and cold start problems. Users' social data such as friends, trusts, etc. increase the quality and confidence of results. Finally, using gossip protocols increases the scalability and the performance of the system [78].

2.7 Conclusion

In this chapter, we introduced the main concepts and approaches of recommendation systems, and their limitations. We also highlighted the requirements that are needed to design P2P recommendation systems. We introduced P2P systems according to three main classes: unstructured, structured and dynamic (using gossip protocols).

Since in this thesis, we exploit techniques from information retrieval and recommendation systems, we reviewed the existing solutions in both areas in the context of P2P systems, namely P2P content management systems and P2P prediction systems. Finally, we summarized our review of related work and our observations.

We pointed out that none of the aforementioned approaches meets all the requirements of recommendation systems. Therefore, our goal in this thesis is to provide a decentralized RS for large-scale data sharing for P2P online communities that satisfies these requirements. Our approach is to leverage users' interests combined with social data, and using gossip protocols to disseminate users' information, construct and maintain the P2P overlay. Users' interests are extracted automatically from their contents and taking into account users' ratings.

Chapter 3 P2Prec: P2P Recommendation for Large-scale Data Sharing

Abstract. In this chapter, we propose a P2P recommendation system called P2Prec that facilitates document sharing for on-line communities. Given a query, the goal of P2Prec is to find relevant peers that can recommend documents that are relevant for the query and are of high quality. A document is relevant to a query if it covers the same topics, and it is of high quality if relevant peers have rated it highly. The topics each peer is interested in are automatically calculated by analyzing the documents the peer holds, and the peer becomes relevant for a topic if it holds a certain number of highly rated documents on this topic. We propose new semantic-based gossip protocols to efficiently disseminate information about peers' topics and relevant peers. In addition, we propose an efficient query routing algorithm that selects the best peers to recommend documents based on the gossip-view entries and query topics. In our experimental evaluation, using the TREC09 dataset, we show that P2Prec has the ability to get reasonable recall with acceptable network traffic.

3.1 Introduction

In this chapter, we propose a P2P recommendation system, called P2Prec, that facilitates document sharing for on-line communities. Our approach leverages content- and collaborative-based filtering recommendation approaches. In most collaborative-based filtering systems, topics of interest are derived based on the users' tagging activities that may lead to ambiguous interpretations. In contrast, in our context of online communities, we exploit the fact that people tend to store high quality content related to their topics of interests. Thus, we can automatically derive the users' topics of interest from the documents they store and the ratings they give, without requiring tagging.

P2Prec works with a set of documents distributed over a large-scale network of volunteer and autonomous peers (users) willing to share and rate their documents. It automatically extracts the topics a user is interested in by relying on a generic automatic topic classifier such as Latent Dirichlet Allocation (LDA) [19]. LDA uses Bayesian statistics and machine learning techniques to infer to the hidden topics in unlabeled content (documents, collections of images, music, DNA sequences, etc.) from labeled content whose topics have already been determined.

To guide recommendation and manage sparsity, we propose a metric to identify the relevance of a user with respect to a given topic. That is, a user is considered *relevant* to give recommendations for a specific topic *t* if it has a sufficient number of highly rated documents related to *t.*

Information about the interest and relevance of users is disseminated over the P2Prec overlay using gossip protocols. With *random gossiping* [50][69], each user keeps locally a view of its dynamic acquaintances (or view entries), and their corresponding topics of interest. Periodically, each user chooses randomly a contact (view entry) to gossip with. The two involved peers 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. Several algorithm parameters, such as the gossip contact, the view subset, etc. are chosen randomly.

In P2Prec, users search for documents that are related to their topics of interests. Thus, in order to increase the quality and the efficiency of recommendation, we propose a *semantic gossip* approach where semantic information, such as user's topics of interest, is taken into account while gossiping. The content of this chapter is mainly based on our material published in [36][38] and has the following contributions.

- We propose a new approach for decentralized recommendation that leverages collaborative- and content-based filtering recommendation approaches. To guide recommendation, we introduce the concept of *relevant users*.
- We propose a P2Prec overlay that enables efficient decentralized recommendation using gossip protocols. We propose two new semantic-based gossip protocols that take into account semantic information such as the users' topics of interest and user relevance, while maintaining the nice properties of gossiping.
- We propose an efficient query routing algorithm that takes into account the most relevant view entries, and recommends the best users to provide recommendation for a query. We use information retrieval techniques, such as *cosine similarity*, to help P2Prec find relevant documents at each involved peer.
- To rank recommendations at the query initiator, we propose a rank method that takes into account similarities, ratings and document popularity.
- We provide an experimental evaluation using the TREC09 dataset [127] that demonstrates the efficiency of P2Prec.

The rest of this chapter is organized as follows. Section 3.2 defines the problem. Section 3.3 introduces P2Prec basic concepts such as topics of interest and relevant users. Section 3.4 describes how the P2Prec overlay is constructed and maintained via gossip protocols. Sections 3.5-3.6 describe two new gossip protocols, semantic and semantic two-layered gossip, respectively. Section 3.7 describes our solution for query routing and recommendation ranking. Section 3.8 gives an experimental evaluation. Section 3.9 concludes.

3.2 Problem Definition

Intuitively, given a query, we want to recommend the most relevant and qualitative documents from a huge distributed content base. Most recommender systems for web data are centralized and are either content-based or use collaborative-based filtering. Content-based filtering recommends to a user *u* items that are similar to *u*'s previously rated items as described in Section 2.2.2. Collaborative-based filtering, in contrast, recommends to *u* items that have been rated by users who share similar interests with *u* as described in Section 2.2.1. Relying on both content-based and collaborative-based filtering approach, we extract a user's topics of interest based on the documents stored at the user.

Our recommendation model assumes a set *D* of shared documents and a set *U* of users $u_1, \ldots u_n$ corresponding to autonomous peers $p_1, \ldots p_n$. Notice that documents may be replicated as a result of using P2Prec. Thus, each document $doc \in D$ can have many read-only copies. Since we focus on on-line communities, we safely assume that users are willing to rate the documents they store. That is, each document *doc* that has a copy at user *u* has high probability to be rated by *u*. Furthermore, we assume a set *T* of topics. Our system will automatically associate each user $u \in U$ with a set of topics of interest $T_u \subset T$, and a set of relevant topics $T_u^r \subset T_u$ depending on the documents *u* maintains locally and the ratings he/she has given to these documents. More specifically, a topic *t* is of interest for user *u*, i.e, $t \in T_u$, if a specific percentage of *u*'s local documents D_u are related to topic *t* with high probability and are highly rated by *u*. User *u* is considered a *relevant user* for topic $t \in T_u^r \subset T_u$, if *u* is interested in *t* and has a sufficient number of highly rated documents that are related to *t* with high probability, and *u* will be able to provide high quality recommendations related to *t*.

Finally, queries are expressed through keywords and a response to a query *q* is a recommendation defined as:

$$recommendation_q = rank(rec_q^{v1}(doc_1), \ldots, rec_q^{vi}(doc_j)) \qquad (3.1)$$

Different recommendations $rec_q^{v1}(doc)$, $rec_q^{v2}(doc)$, ... may be given for the replicas of a *document doc.* Each *rec* is defined in terms of the similarity between the query *q and* doc_i, and the document popularity. Finally, the rank function may be standard or user defined.

Problem Statement: Given a keyword query *q* and our recommendation model above, the problem we address is how to efficiently retrieve the most relevant users (or peers) to compute $recommendation_q$ and selectively choose the best recommendations.

3.3 P2Prec Basic Concepts

In this section, we introduce P2Prec basic concepts for managing topics of interests and relevant users. First we present how topics are extracted from a set of documents by using LDA. Next, we introduce how we extract users' topics of interests from documents they store, and how we define the concept of relevant users.

3.3.1 Topics Extraction

In P2Prec, topics are automatically extracted from a set of documents to produce the set of topics T, and for each user $u \in U$, its set of topics of interest $T_u \subset T$. 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 literature [26] to tackle this problem. We use, LDA, 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 a mixture of words from the topics *food* and *cooking*.

LDA assumes a fixed finite number of topics. Hierarchical Dirichlet Processes [151] has been proposed as an extension to the standard LDA [19] to adapt the number of topics automatically but is more complex. In P2Prec, we use the standard LDA and thus use a fixed finite number of topics. We choose LDA because it is efficient in clustering high dimensional and sparse data, and it has ability to solve synonymy (different words with identical meaning) and polysemy (same word with multiple meanings). But we could easily extend P2Prec to use Hierarchical Dirichlet Processes.

We adapt LDA for P2Prec to proceed in two steps: the training (at the global level, see Figure 3.1(a)), and inference (at the local level, see Figure 3.1(b)). Training, denoted by *Global-Training-LDA*, 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 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 and produces a set $B=\{b_1,.. b_d\}$ of bags. Each bag $b \in B$ is tagged with a label t (we refer to it as topic t). The set 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 peers and estimates ϕ. Each version of ϕ is attached with a timestamp value.

The inference part of LDA, denoted by *Local-Inference-LDA*, is performed locally at each (peer) 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, when-

ever 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 produces 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*.

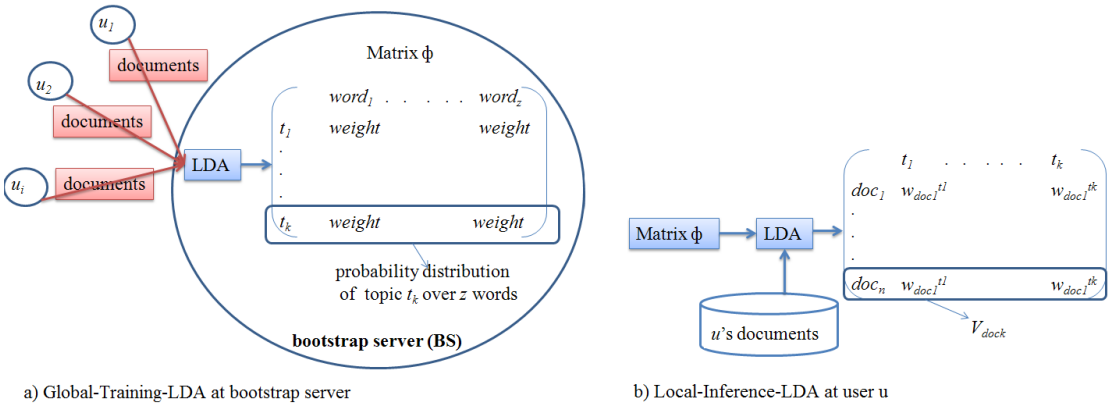


Figure 3.1. LDA under P2Prec context

3.3.2 Topics of Interest and Relevant Users

Now introduce the concepts of users' topics of interest and relevant users necessary to guide recommendation. Algorithm 3.1 illustrates how each user computes its topics of interests with which we determine relevant users. Given a set of documents D_u stored locally at user *u*, we extract the topics of interest $T_u \subset T$ in two steps. First, we compute the document quality for each document $doc \in D_u$ that user *u* has rated and we record the quality locally in a vector *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 the *Local-Inference-LDA*, by the rate $rate_{doc}^{u}$ that has been given by user *u* for *doc*. Thus, we have:

$$quality(doc,u) = [w_{doc}^{t_1} * rate_{doc}^{u} \ldots\ldots. w_{doc}^{t_d} * rate_{doc}^{u}] \text{ (corresponds to line 2)}$$

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*, denoted by $relate_t(doc,u)$, 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.,

$$relate_t(doc,u) = \begin{cases} 1, & w_{doc}^{t} * rate_{doc}^{u} \geq threshold \\ 0, & otherwise \end{cases} \quad (3.2)$$

Topic(s) related to a document leverages the user's rating and document's semantic content. If a document has not been rated explicitly by user *u*, we still have the ability to compute the topics that are related to it. In this case, we consider the document to be highly related to topic *t* if its weight in that topic exceeds a threshold value.

In the second step (lines 3, 4 and 5), user u counts how many documents are highly related to each topic $t \in T$. The number of documents that are highly related to $t \in T$ represents u's degree of interest in that topic, denoted by $degree_u^t$, i.e.,

$$degree_u^t = \sum_{i=1}^{|D_u|} relate_t(doc_i, u) \quad (3.3)$$

Then, user u implicitly computes its topics of interest $T_u \subset T$ (lines 9, 10 and 11). User u is considered interested in topic t if a percentage y (or absolute value) of its local documents D_u are highly related to topic t, i.e.,

$$t \in T_u \quad \text{if} \quad \frac{degree_u^t}{|D_u|} \geq y \quad (3.4)$$

Furthermore, user u is considered a relevant user for topic t that belongs to its relevant topics T_u^r i.e., $t \in T_u^r$, if u is interested in t and $degree_u{}^t$ exceeds a number x (which is system defined), i.e.,

$$t \in T_u^r \quad \text{if} \quad degree_u^t \geq x \quad \text{and} \quad t \in T_u \quad (3.5)$$

In other words, u is considered a relevant user in topic $t \in T_u^r$ if it is interested in t and has a sufficient amount of documents that are highly related to topic $t \in T_u^r$ (lines 12 and 13). Otherwise u is not a relevant user for topic t even though u might be interested in t. We denote a user that is not relevant for any topic as a *non-relevant user*.

User u has the ability to download and rate the documents it receives, and add or delete documents. Thus, its relevance (topics of interest) may change over time. To capture this dynamic behavior, user u computes its topics of interest T_u and relevant topics T_u^r periodically, or if a number of documents have been added to (or deleted from) its D_u and exceeds a system-defined threshold.

Algorithm 3.1- Compute-Topics-Of-Interest(V_{doc}, $rate_{doc}$)

Input: user u's document topic vectors, V_{doc} where $doc \in D_u$; user u's document rating, $rate_{doc}^{u}$ where $doc \in D_u$

Output: user u's topics of interest T_u

```
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 relate_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       u add t to T_u
12       If degree_u^t ≥ x and t ∈ T_u then
13          u add t to T_u^r
14       End If
15    End If
16 End For
```

3.4 P2Prec Overlay

In this section, we first describe how the P2Prec overlay is constructed and maintained via gossip protocols. Then, we introduce our query routing solution. Finally, we describe the well-known random gossip protocol [50][69], and discuss its limitations for P2Prec

3.4.1 Overlay Construction

In P2Prec, users use gossip protocols to construct the P2Prec overlay and exchange a subset of their views in an epidemic manner [5]. Users also gossip to detect failed users. Gossip protocols [50][69] have attracted a lot of interest for building and managing unstructured networks. With gossip, each user periodically (with a gossip period denoted by C_{gossip}) exchanges a subset of its state, called *local-view*, with another user. Thus, after a while, as with gossiping in real life, each user will have a partial view of the other users in the system.

- Each user's *local-view* contains a fixed number of entries, denoted by *view-size*. Each entry refers to a user *u*, and contains *u*'s gossip information such as:
- *u*'s IP address;
- *u*'s topics of interest T_u, each topic $t \in T_u$ being associated with a Boolean field that indicates whether *u* is relevant in that topic.

Users' topics of interest and relevant users' information are disseminated using gossip protocols in order to guide queries for recommendation retrieval. When a user is interested in a topic *t* it may be a candidate to serve a query on topic *t.* This corresponds to the case in which there is no relevant user on that topic in the view.

In the example of Figure 3.2, *u* carries in its *local-view* two users v_1 and v_2. User v_1 is interested in two topics t_1 and t_2. Figure 3.2 shows that v_1 is not relevant either in t_1 or t_2. As user v_1 is not relevant in any topic, then v_1 is a non-relevant user. User v_2 is interested in two topics t_1 and t_2. Figure 3.2 shows that v_2 is relevant in t_2, and not relevant in t_1. As v_1 is relevant at least in one topic, then v_2 is a relevant user.

$T_u = <t_1, t_2>$

User	T_{vi}
v_1	$(t_1,0);(t_2,0)$
v_2	$(t_1,0);(t_2,0)$

Figure 3.2. User *u*'s *local-view*

We choose gossip protocols for the following reasons. First, the continuous exchange of subsets of *local-views* between users enables the building of an unstructured overlay network in a continuous manner, which reflects the natural dynamism of P2P networks and helps providing very good connectivity despite failures or peer disconnections [50]. Second, gossiping provides a reliable way to disseminate infor-

mation in large-scale dynamic networks, so that users discover new users [45]. Third, it 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 [50]. Finally, it is scalable, reliable, efficient and easy to deploy [66].

3.4.2 Query Processing

Whenever a user submits a query, *local-view* is used as a directory to redirect the query to the appropriate relevant users. 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, and V_q is query q's topic vector. Notice that q's topic vector V_q is computed using the *Local-Inference-LDA*. T_q represent q's topics and u corresponds to the address of q's initiator. When a user u initiates a query q, it routes q as follows: first, it extracts q's topic vector V_q using the *Local-Inference-LDA*. Then, user u computes q's topics T_q from q's topic vector V_q. The query 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). Then, user u uses its *local-view* to find relevant users that can give recommendation for q's topics T_q, and then redirects q to those relevant users after reducing TTL.

Whenever a user u receives a query q that has been initiated by a user v, it returns to q's initiator the recommendation information it has which are related to q, and recursively selects from its *local-view* the relevant users in q' topics T_q. Afterwards, u redirects q to those relevant users as long as TTL does not reach zero. More details on query processing are given in Section 3.7.

3.4.3 Random Gossip Protocol

The basic random gossip protocol (Rand for short) proceeds as follows: a user u (either relevant or non-relevant) acquires its initial *local-view* during the join process using a bootstrap technique. We register each user that has joined P2Prec at a bootstrap server. Whenever a user u joins the system, it randomly selects a set of users from the bootstrap server to initialize its *local-view*. Notice that u's *local-view* may carry relevant and non-relevant users.

Whenever a user u initiates an information exchange, it selects a random contact v from its *local-view* to gossip with. Then, u selects a random subset of size L_{gossip} -1, denoted by *viewSubset*, from its *local-view*, and includes itself into *viewSubset*. Then, u sends *viewSubset* to v. Similarly, 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, u searches for a recommendation, it uses its *local-*

view to identify the relevant users in its view that can provide recommendation for the query.

Rand does not take into account user *u*'s topics of interests during the gossip exchanges. This reduces the possibility of having users in *u*'s *local-view* which are similar to *u* in terms of topics of interest, which reduces the possibility of getting better responses. In particular, the exchange does not consider whether the view contains users that are relevant for the topics user *u* is interested in. In the following we refer to this as similarity. For now, we informally assume two users are similar if they have similar interests. In Section 3.5.2 we provide a formal definition.

In the following we present three examples where Rand limits the quality of the gossip exchange:

1. Consider two users *u* and v_1 that are not similar, and user v_1 is in *u*'s *local-view* (see Figure 3.3(a)), because they do not have any common topic of interest. Let us suppose that v_1 has several users in its *local-view* that are similar to v_1. For instance v_4 is in v_1's *local-view*, and has topics of interest T_{v4} which are the same as v_1 topics of interest T_{v1}. By transitivity these users are not similar to *u*. If *u* chooses v_1 as a gossip contact, with high probability it will end by filling its *local-view* with un-similar users (see Figure 3.3(b)), because most of the users in *u*'s *local-view* do not have topic in common with *u*'s topics of interest. In the example of Figure 3.3, *u* selects v_1 to gossip with, and sends to v_1 $ViewSubset_u$ which, in addition to itself, includes v_2 and v_3. Similarly, v_1 returns to *u* a $vewSubset_{v1}$ which includes in addition to itself users v_5 and v_6. Once *u* receives the $viewSubset_{v1}$, it merges $viewSubset_{v1}$ with its *local-view* in a buffer, and then updates its *local-view*.

$T_u = \langle t_1, t_2 \rangle$

User	*topics*
v_1	$(t_3,1);(t_4,1)$
v_2	$(t_1,1);(t_2,1)$
v_3	$(t_2,1);(t_3,0)$

$T_{v1} = \langle t_3, t_4 \rangle$

User	*topics*
v_4	$(t_3,1);(t_4,1)$
v_5	$(t_3,1);(t_5,0)$
v_6	$(t_4,1)$

a) *u* and v_1 *local-views* before gossip

$T_u = \langle t_1, t_2 \rangle$

User	*topics*
v_5	$(t_3,1);(t_5,0)$
v_2	$(t_1,0);(t_2,0)$
v_6	$(t_4,1)$

$T_{v1} = \langle t_3, t_4 \rangle$

User	*topics*
u	$(t_1,1);(t_2,1)$
v_2	$(t_1,1);(t_2,1)$
v_6	$(t_4,1)$

b) *u* and v_1 *local-views* after gossip

Figure 3.3. Users *u* and *v* are not similar

2. Consider now that *u* and *v* are similar (see Figure 3.4(a)), because *u*'s topics of interest T_u and v_1's topics of interest T_{v1} are similar. However, v_1 has many un-similar users in its *local-view*. For instance v_4 is in v_1's *local-view* and does not have any topic of interest that v_1 is interested in. By transitivity, these users are not similar to *u*. If *u* chooses *v* as gossip contact again, with high probability it will end up filling its *local-view* with un-similar users (see Figure 3.4(b)), because most of the users in *u*'s *local-view* do not have topic in common with *u*'s topics of interest. In the example of Figure 3.4, *u* selects v_1 to gossip with, and sends to v_1 $ViewSubset_u$ which includes in addition to itself users v_2 and v_3. Similarly, v_1 returns to *u* a $vewSubset_{v1}$ which includes in addition to itself users v_5 and v_6. Once *u*

receives the $viewSubset_{v1}$, it merges $viewSubset_{v1}$ with its *local-view* in a buffer, and then updates its *local-view*.

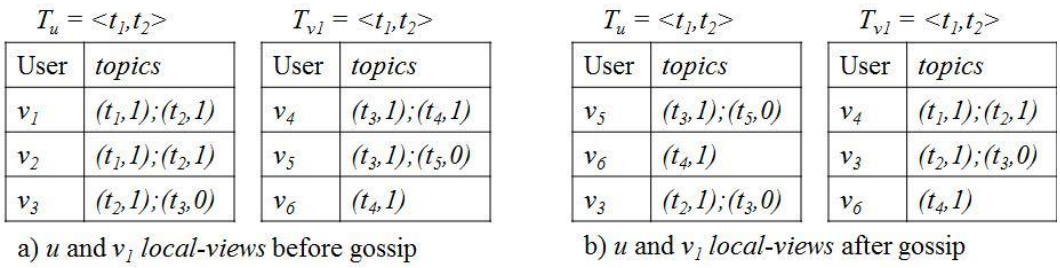


Figure 3.4. User *u* and *v* are similar

3. Consider the case that several users $u_1,\ldots,u_k$ are non-relevant users, and *u*'s *local-view* carries mostly non-relevant users (see Figure 3.5(a)), for example v_2 is in *u*'s *local-view* and is not relevant in any topic. In this case, the gossip exchanges are useless for serving queries (see Figure 3.5(b)). For example, after gossiping, *u* does not carry in its *local-view* any user that is relevant in topic $t_1 \in T_u$.

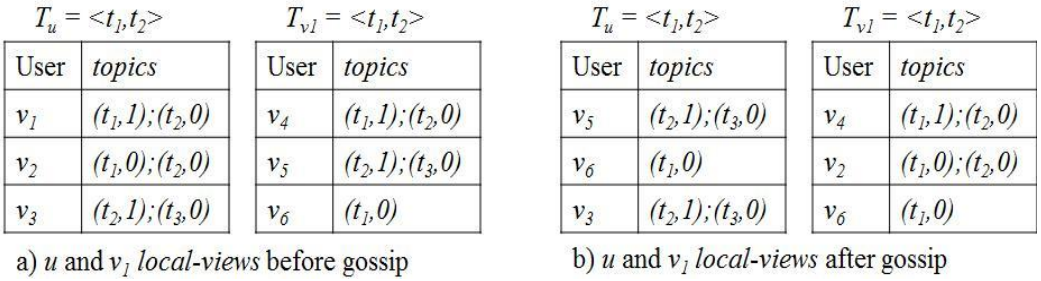


Figure 3.5. User *u* and *v* carry mostly non-relevant users

Based on the above examples, we conclude that Rand may generate uninteresting view states resulting in low query responses.

3.5 Semantic Gossiping

In this section, as a first approach to Rand's limitations, we present a new semantic gossip protocol (called Semt). The goal is to selectively maximize the number of relevant users at each user *u*'s *local-view* that are similar to *u*. First, we give our criteria for keeping similar relevant users in the *local-views*. Then, we present in detail the active and passive behavior of Semt.

Recall that our objective is to improve the efficiency of returning useful recommendations for on-line communities. We let each user *u* maintain a *local-view* of rele-

vant users similar to *u*. Thus, when *u* initiates a query *q* (see Algorithm 3.4), it searches for a relevant user *v*∈*u's local-view* so that *v* can give recommendation for *q*. If *u* finds such relevant user, then *u*'s *hit-ratio* is increased. *Hit-ratio* is defined as the percentage of the number of queries that have been answered. Moreover, *u* likes to find many relevant users in its *local-view* that can serve its queries, and this reduces query response time (time spent to retrieve useful recommendations).

To measure user *u*'s *hit-ratio*, we use a *query-history* that keeps the track of past queries. With Semt, when a user *u* chooses a contact, it selects a user *v* that has high *hit-ratio*, and is similar to user *u*. For that, *u* includes into *viewSubset* the relevant users that are similar to *u*, and have high *hit-ratios*. Note that *hit-ratio* can be easily added as an attribute of a *local-view* entry, and becomes part of the gossip message.

In the rest of this section, we present our techniques to compute *hit-ratio*, and the similarity functions.

3.5.1 Computing the Hit-Ratio

To compute a user's *hit-ratio*, 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 $|H_u|$. H_u contains a set of entries, each entry referring to a past query *q* that *u* has initiated. Each past query *q* entry included in H_u contains *q's* topics T_q and its query state s_q. Query state s_q can be either 1 or 0. The value of 1 for s_q denotes a *query-success*, i.e., there was at least one relevant user in *u*'s *local-view* that was able to serve query *q*. In contrast, $s_q = 0$ denotes a *query-fail*, i.e., user *u* has not found any relevant user in its *local-view* that can give recommendations for query *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:

$$hit\text{-}ratio_u = \frac{\sum_{i=1}^{n} s_{q_i} \in H_u}{|H_u|} \qquad if\ s_{q_i} = 1 \tag{3.6}$$

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

3.5.2 Similarity Functions

Recall that each user has a set of topics of interest, and each relevant user *v* a set of relevant topics. Thus, we measure the similarity between a user *u* and a relevant user *v*, denoted by *distant(u,v)*, by counting the overlap between *u*'s topic of interests T_u and *v*'s relevant topics T_v^r. We use the Dice coefficient [35] which is:

$$distant(u, v) = \frac{2|T_u \cap T_v^r|}{|T_u| + |T_v^r|} \tag{3.7}$$

We could also use other similarity functions such as cosine, jaccard, etc. Similarly, we use the Dice coefficient to measure the similarity between a query q and a relevant user v:

$$distant(q,v) = \frac{2|T_q \cap T_v^r|}{|T_q| + |T_v^r|} \quad (3.8)$$

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

3.5.3 Semantic Gossip Behaviors

The behavior of Semt at a user u is illustrated in Algorithm 3.2. The active behavior describes how u initiates a periodic gossip exchange message, while the passive behavior shows how u reacts to a gossip exchange initiated by some other user v. Each user u acquires its initial *local-view* during the join process using a bootstrap technique. We register each relevant user which has joined the P2Prec at a bootstrap server. Whenever a user u joins the system, it selects randomly a set of relevant users from the bootstrap server to initialize its *local-view.* Notice that u's *local-view* only carries relevant users.

The active behavior is executed every time unit C_{gossip}. A user u initiates a communication message and computes the similarity distance between itself and each relevant user v in its *local-view* (line 4). Then, u computes the rank of each relevant user v in its *local-view*, denoted by *rank(v).* A relevant user v's rank at user u depends on the similarity distance between u and v, and v's *hit-ratio* if v has issued more than z number (where z is system defined) of queries within an interval of time i.e., $H_v \geq z$. Otherwise v's rank depends on similarity distance between u and v only. Accordingly the *rank(v)* is:

$$rank(v) = \begin{cases} distant(u,v) & if\ |H_v| \leq z \\ hit\text{-}ratio_v * distant(u,v) & otherwise \end{cases} \quad (3.9)$$

Usually z is very small, that is to prevent the relevant users that are similar to u, but do not issue queries from getting very low ranks. Note that $|H_v|$ can be easily added as an attribute of a *local-view* entry, and becomes part of the gossip message.

Once u has computed the rank of each relevant user v *rank(v)* in its *local-view*, adds *rank(v)* to a *RankList* (lines 5 to 10) which contains the relevant users' entries along with their ranks. Once u has computed the relevant users' ranks and added them in the *RankList*, it selects from the *RankList* a relevant user v which has the highest rank to gossip with, using the **selectTop**() method (line 12). The relevant user v with the highest rank is the relevant user that is most similar to u and has the highest *hit-ratio*$_v$.

Once user u has selected a relevant user v to gossip with, it selects L_{gossip} entries from the *RankList* which have the highest rank using **SelectTopEntries**() (line 13). These entries compose user u *viewSubset.* After that user u sends to v the *viewSubset* (line 14).

In turn, user *u* will receive a *viewSubset** of user *v*'s *local-view* (line 15). Upon receiving *viewSubset**, *u* computes the rank for each relevant user *v* in *viewSubset** and adds it to the *RankList* (lines 16-24). Recall that *RankList* includes also the rank of the relevant users at *u*'s current *local-view*. Then, the method **SelectTopEntries**() selects *view-size* entries from the *RankList* which have the highest rank to become the new *local-view* (line 25).

In the passive behavior, a user *u* waits for a gossip message from a user *v*. Upon receiving a message (line 3), it computes the rank of the relevant users in its *local-view* (lines 4-11). Then, it uses **SelectTopEntries**() to select *viewSubset** of L_{gossip} entries from the *RankList* that have the highest rank (line 12). Then, it sends back *viewSubset** to user *v*. Then, it computes the rank of the relevant users in the received *viewSubset* (lines 14-22). Finally, it updates its *local-view* by selecting *view-size* entries from the *RankList* that have the highest rank.

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* remains the same contact for long period of time. To increase the randomness and prevent user *u* from selecting the same contact *v* for a long period of time, each user *u* stores in a list *L*, the last ℓ recent contacts that have been selected for gossiping. Then, instead of blindly selecting the top ranked relevant user in *RankList*, to gossip with it selects the first user in *RankList* that is not in the list *L* of users with whom *u* has recently gossiped.

Furthermore, the fact that *viewSubset* and the gossip contact are not chosen randomly may reduce the user's ability to discover new relevant users.To overcome this limitation, we propose a semantic two-layered gossiping (Section 3.6).

Algorithm 3.2- Gossiping($local\text{-}view_u$)

```
// Active behavior
Input: local-view_u
Output: updated local-view_u
1  Forever do
2    wait(C_gossip)
3    For each relevant user v ∈ local-view_u do
4      user u computes distant(u,v)
5      If |H_v| ≥ z then
6        rank(v) = distant(u,v)
7      Else
8        rank(v) = hit-ratio_v * distant(u,v)
9      End If
10     user u adds <rank(v) ,v> to RankList
11   End For
12   user v = selectTop(RankList)
13   viewSubset = SelectTopEntries(RankList,L_goosip)
14   User u send <viewSubset > to user v
15   User u receive viewSubset * from user v
16   For each relevant user v ∈ viewSubset * do
17     user u computes distant(u,v)
18     If |H_v| ≥ z then
19       rank(v) = distant(u,v)
20     Else
21       rank(v) = hit-ratio_v * distant(u,v)
22     End If
23     user u adds <rank(v) ,v> to RankList
24   End For
25   Local-view_u =SelectTopEntries(RankList, view-size)

//Passive behavior
Input: viewSubset of a user v; local-view_u
Output: updated local-view_u
1  Forever do
2    waitGossipMessage( )
3    receive <viewSubset > from user v
4    For each relevant user v∈u's local-view do
5      user u computes distant(u,v)
6      If |H_v| ≥ z then
7        rank(v) = distant(u,v)
8      Else
9        rank(v) = hit-ratio_v * distant(u,v)
10     End If
11   End For
12   viewSubset* = SelectTopEntries(RankList,L_goosip)
13   send viewSubset* to user v
14   For each relevant user v ∈ viewSubset do
15     user u computes distant(u,v)
16     If |H_v| ≥ z then
17       rank(v) = distant(u,v)
18     Else
19       rank(v) = hit-ratio_v * distant(u,v)
20     End If
21     user u adds <rank(v) ,v> to RankList
22   End For
23   Local-view_u = SelectTopEntries(RankList, view-size)
```

3.6 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 of Semt. Rand preserves gossiping properties and gives users the ability to discover new relevant users. These new relevant users are then taken into account in Semt to find new similar relevant users.

2LG uses the following approach. 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) 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 (as described in Section 3.4.3). Then, it initializes its *semantic-view* by computing the ranks of the relevant users in its initial *random-view* and selects S_{size} entries which have the highest ranks. Then, *u* periodically (with a gossip period C_{random} and $C_{semantic}$) performs Rand and Semt asynchronously. Notice that $C_{semantic} >> T_{random}$ because user semantics (topic of interests) are not changed rapidly. But we assume that C_{random} is small enough to capture the dynamicity of the network, as peer joins and leaves keep happening continuously.

In 2LG, we adopt Semt (see Algorithm 3.2) with a modification to its active behavior only, to take advantage of the *random-view*. Algorithm 3.3 shows the modifications on the active behavior of Semt for 2LG. In principle, the lines 1-24 of Algorithm 3.2 do not change except that user *u* uses its *semantic-view* and not the *local-view* for creating the *RankList*. Thus, these lines are not repeated in Algorithm 3.3. However, line 25 of Algorithm 3.2 is replaced by the steps taken in Algorithm 3.3. After line 16 of Algorithm 3.2, the *RankList* includes the rank of the relevant users at *u*'s current *semantic-view* and the ranks of the relevant users in the *viewSubset* that *u* has received during the exchange. From there, and different to Semt, 2LG also takes into account the relevant users in its *random-view* as follows: *u* ranks the relevant users in its *random-view*, and adds them to the *RankList* (lines 1-9 in Algorithm 3.3). Then, *u* selects the S_{size} entries from *RankList* that have the highest rank to be its new *semantic-view* (line 10 of Algorithm 3.3).

In the example of Figure 3.6, we show the framework of 2LG at user *u*. User *u* performs Rand and Semt asynchronously. It performs Rand as described in Section 3.4.3: it selects randomly a user v_1 from its *random-view* to gossip. Then it selects randomly a $viewSubset_u$ from its *random-view* and sends it to v_1. Afterwards, user *u* receives a $viewSubset_{v1}$ from v_1. Once *u* has received $viewSubset_{v1}$, it updates its *local-view*, by merging $viewSubset_{v1}$ with its current *random-view* in a buffer, selecting R_{size} entries randomly, and updates its *random-view*.

User *u* performs Semt as described in Algorithm 3.2 with the modification of Algorithm 3.3. It computes the rank of each relevant user *v* in its *semantic-view* and adds them to *RankList*. Then it selects the relevant user v_2 that has the highest rank to gossip with. Then it selects a $viewSubset_u$ from the *RankList* that have the highest rank and sends it to v_2. Afterward, *u* receives a $viewSubset_{v2}$ from v_2. Once user *u* has received, $viewSubset_{v2}$, it updates its *semantic-view* as follows: 1) It computes the rank of each relevant user *v* in the $viewSubset_{v2}$ and adds it to the *RankList*. 2) It computes

the rank of each relevant user *v* in its *random-view* and adds to the *RankList*. 3) It selects S_{size} entries from the *RankList* that have the highest rank.

Algorithm 3.3. Modifications on the active behavior of Semt for 2LG

Input: *semantic-view*$_u$; *random-view*$_u$
Output: updated *semantic-view*$_u$
Lines 1-24 of the active behavior of Algorithm 3.2

```
1   For each relevant user v∈random-view_u do
2       user u computes distant(u,v)
3       If |H_v| ≥ z then
4           rank(v) = distant(u,v)
5       Else
6           rank(v) = hit-ratio_v * distant(u,v)
7       End If
8       user u adds <rank(v),v> to RankList
9   End For
10  semantic-view_u = SelctTopEntries(RankList,Ssize)
```

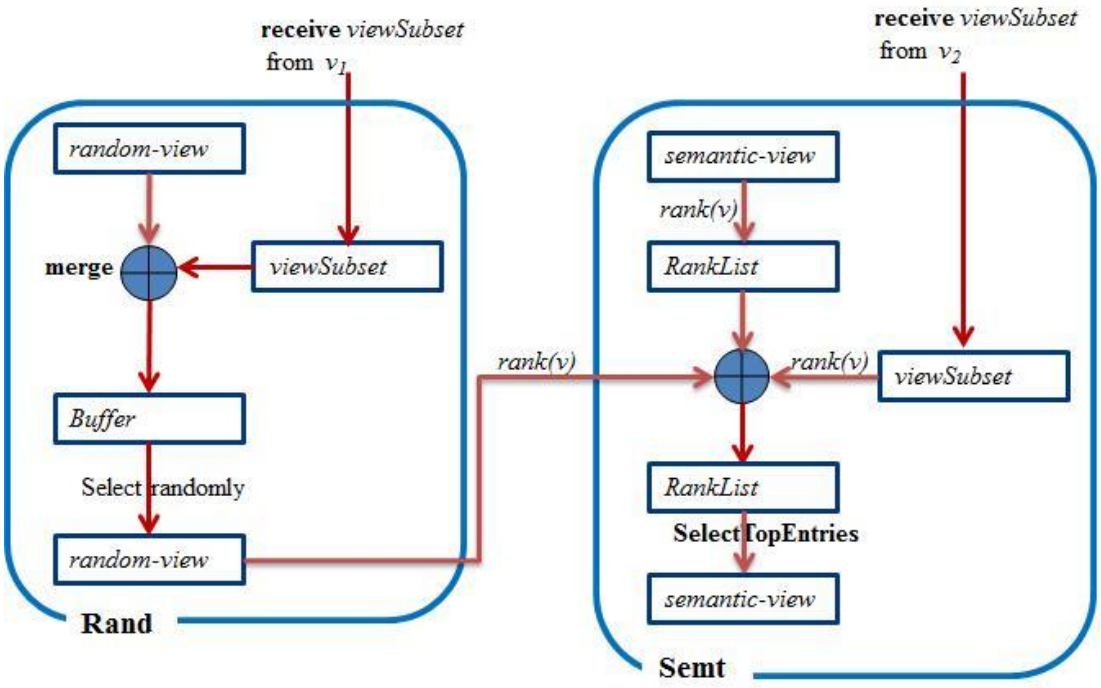


Figure 3.6. The 2LG framework at user *u*

3.7 Query Routing and Recommendation 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 recommendations. Finally, we show how users can manage query failures.

3.7.1 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* [75]. 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 interest, such as the most representative words in topics of interest. In the pull type, user u issues a query q with keywords. For both types, the system extracts q's topic vector, denoted by $V_q = [w_q^{t_1} ....w_q^{t_d}]$, using LDA as we did for a document. Then query topic(s) $T_q \subset T$ are extracted as described in Section 3.4.2.

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 3.4 illustrates the behavior of query processing of each user u. In active behavior, u issues a query q and proceeds as follows. First, it selects from its *local-view* the relevant users that are similar to q in terms of topics. Then, it redirects q to those relevant users after reducing the query TTL by one (lines 1 to 6). In other words, user u selects each relevant user $v \in u$*'s local-view* that are similar to q, i.e., $distant(q,v) \neq 0$, and then redirects q to them. If 2LG is used, *u's local-view* is the union of its *random-view* and its *semantic-view*.

If user u does not find any relevant user v in its *local-view* that is similar to q, the query q is considered failed, and u uses the *query-history* of the users in its *local-view* to support q (lines 7 to 14) (presented in Section 3.7.3). Once user u receives the recommendation information from the responders, it ranks those recommendations based on their popularity and semantic similarity (lines 15 to17) (presented in Section 3.7.2).

In the passive behavior, when user u receives a query q, it processes q as follows. First, u selects from its *local-view* the relevant users that are similar to q, and redirects the query to them if the query's TTL is not yet zero (lines 9 to 16). Second, user u measures the similarity between query q and each document user u has locally (lines 3 and 4). The similarity between a document *doc* and q, denoted by $sim(doc,q)$, is measured by using the cosine similarity [130] 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:

$$sim(doc, q) = \frac{\sum_{i=1}^{d} w_q^{t_i} * w_{doc}^{t_i}}{\sqrt[2]{\sum_{i=1}^{d} w_q^{t_i} * w_q^{t_i} * \sum_{i=1}^{d} w_{doc}^{t_i} * w_{doc}^{t_i}}} \quad (3.10)$$

Finally, u returns to the query initiator the recommendations for the documents whose similarity exceeds a given (system-defined) threshold (lines 5 and 6).

With such query routing, we avoid sending q to all neighbors, thus minimizing the number of messages and network traffic for q.

Algorithm 3.4- Query Processing

//Active behavior: **Route-Query**(*q, local-view*$_u$)

Input: query *q* (*word*$_i$, TTL, V_q, T_q,*u*); *local-view*$_u$
Output: submit *q* to potential relevant users

```
1   For each relevant user v∈local-view_u do
2     If distant(q,v) ≠0 then
3       u send q to v
4       q.TTL = q.TTL-1
5     End if
6   End For
7   If query-fail  then
8     For each user v ∈ local-view_u  do
9       user u retrieve user v query-history H_v
10        If  distan(q,q_i) ≠ 0 and s_qi=1 s.t. q_j ∈H_v then
11          User u Send q  to user v
12        End If
13      End For
14  End If
15  If user u Receives rec_1,…, rec_n then
16    User u Ranks (rec_1,…, rec_n)
17  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 information recommendations for query *q; u* send *q* to potential relevant users

```
1   Forever do
2   Receive query q
3   For each doc∈D_u  do
4     Sim(q,doc) = CosineSimilarity(V_q,V_doc)
5     If Sim(q,doc) greater than threshold then
6       recommend doc  to q's initiator
7     End If
8   End For
9   If q.TTL not equal to zero then
10    For each relevant user v∈local-view_u do
11      If distant(q,v) ≠0 then
12        u send q to v
13        TTL = TTL-1
14      End if
15    End For
16  End If
```

3.7.2 Ranking Recommendations

Assume the query initiator receives $rec_q^{v1}(doc_1), \ldots, rec_q^{vi}(doc_j)$ from the responders, where $rec_q^{v}(doc)$ is the recommendation that has been given for a document *doc* from a responder *v*. $rec_q^{v}(doc)$ includes the similarity between query *q* and document *doc*. With this, the initiator ranks $rec_q^{v1}(doc_1), \ldots, rec_q^{vi}(doc_j)$ to provide $recommendations_q$. The recommencations $rec_q^{v1}(doc_1), \ldots, rec_q^{vi}(doc_j)$ are ranked based on their popularity and semantic similarity (line 16 in the active behavior of Algorithm 3.4). That is, the rank of a $rec_q^{v}(doc)$, denoted by $rank(rec_q^{v}(doc))$, reflects its semantic relevance with *q* and its popularity:

$$rank\left(rec_q^v(doc)\right) = a * sim(doc, q) + b * pop(doc) \qquad (3.11)$$

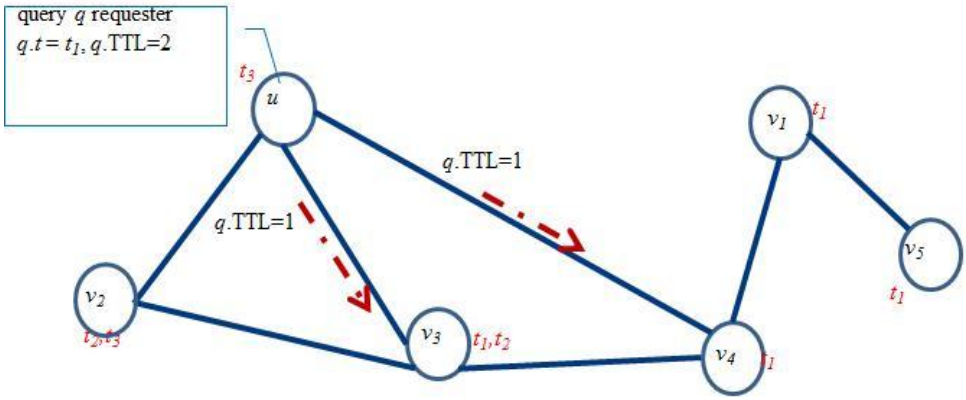
where *a* and *b* are scale parameters such that $a + b = 1$ and $pop(doc)$ is the popularity of *doc*. The popularity is equal to the number of replicas this document has, i.e., the number of users that store a replica of *doc*. The user can specify whether it prefers highly popular documents or documents that are highly semantically relevant by adjusting parameters *a* and *b*. Upon receiving recommendation documents, a user *u* can download a copy of a document, give a rating to it and include it in its document set D_u.

In the example of Figure 3.7, suppose that user *u* initiates a query *q* for topic t_1 with TTL=2. User *u* redirects the query *q* to relevant users v_3 and v_4 after reducing the TTL by 1 (see Figure 3.7(a)).

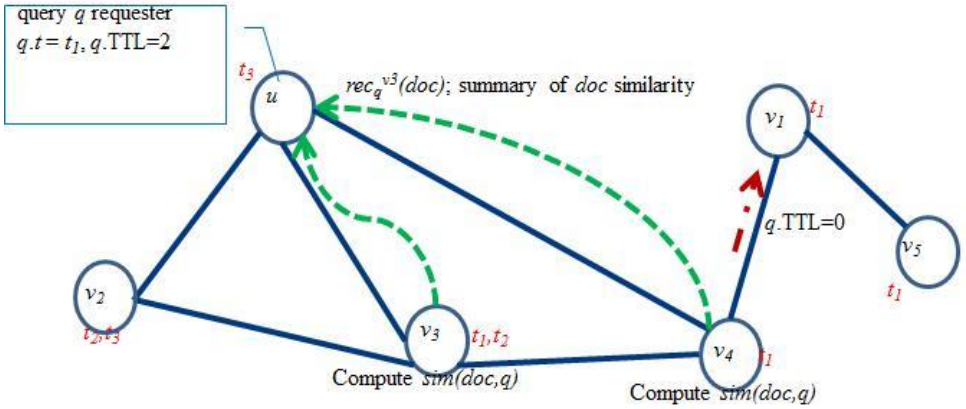
Figure 3.8(b) shows that when v_3 receives *q*, it computes the similarity between *q* and its documents $sim(doc,q)$ where $doc \in D_{v3}$. Then, v_3 returns to *u* the recommendations $rec_q^{v3}(doc)$ for those documents whose similarity exceeds a given threshold. User v_3 stops redirecting *q* even though its TTL is not zero. This is because v_3 does not have a relevant user in its *local-view* that is similar to *q*, and has not yet received a copy of *q*. Similarly user v_4 computes $sim(doc,q)$ for $doc \in D_{v4}$, and returns the recommendations $rec_q^{v4}(doc)$ to *u*. It then redirects *q* to relevant user v_1 after reducing TTL by one (see Figure 3.7(b)).

When user v_1 receives *q*, it computes $sim(doc,q)$ for $doc \in D_{v1}$ and returns the recommendations $rec_q^{v1}(doc_i)$ to *u*. User v_1 does not forward *q* because its TTL has reached zero (see Figure 3.7(c)). Notice that in the case in which *u* does find any relevant user in its *local-view* that can serve *q*, it exploits the *query-histories* of the users in its *local-view*.

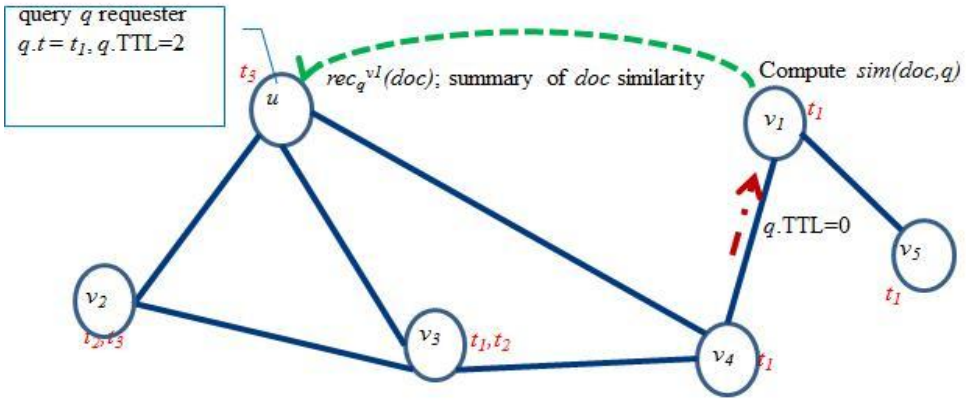
Figure 3.8(d) shows that once the user *u* receives $rec_q^{v}(doc_i)$ from the relevant users v_3, v_4 and v_1, it ranks $rec_q^{vi}(doc_j)$ based on their popularity and semantic similarity to provide $recommendations_q$.



a) u submits and forwards q.



b) Receivers return recommendations to u, and redirect q while q.TTL $\neq 0$.



c) Receiver returns recommendations to u, and stops redirecting q, *because* q.TTL = 0.

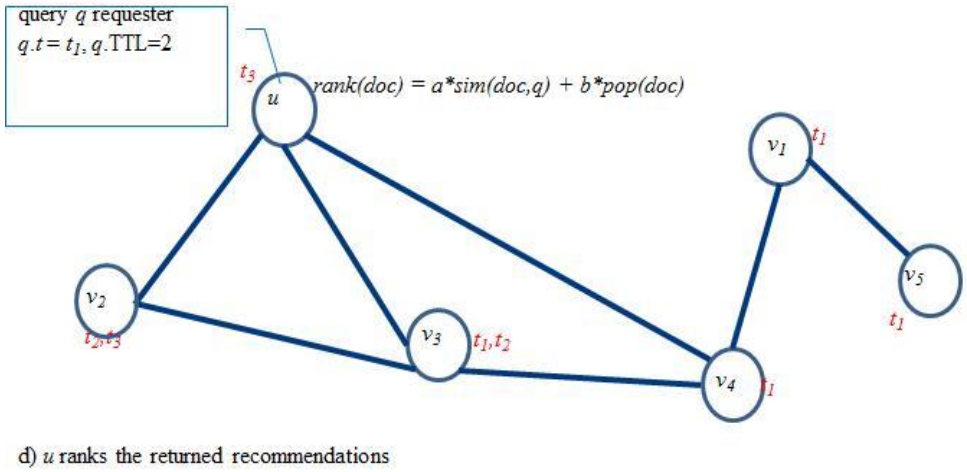


Figure 3.7. The query processing and recommendation ranking

3.7.3 Dealing with Query Failures

We use *query-histories* to support failed queries in the hope to increase the *hit-ratio*. Whenever, a user *u* submits a query *q*, it adds *q*'s topics T_q to its *query-history* along with a state, which indicates if *q* was successfully submitted or not. *q* was successfully submitted, if *u* has relevant user(s) in its *local-view* that are similar to *q*. The idea is that such users can serve other queries that are similar to *q*.

Query *q* is considered as *query-fail* if user *u* does not find any relevant user in its *local-view* that is similar to *q*, i.e., $distant(q,v)= 0$, for each relevant user $v \in local\text{-}view_u$. To handle this situation, we exploit the *query-histories* of the users in *u*'s *local-view*.

Recall that each user *u* maintains a *query-history* H_u. When *u* experiences a *query-fail*, *u* retrieves the *query-history* H_v of each user *v* in its *local-view*. Then, for each H_v, it computes the similarity between *q* and each query $q_i \in H_v$ (lines 8 and 9 in the active behavior of Algorithm 3.4). If there is a query q_i such that $distant(q,q_i)\neq 0$ and $s_{qi}=1$, *u* sends *q* to *v* (lines 10 and 11 in the active behavior of Algorithm 3). Notice that we do not use *query-histories* in passive behavior.

3.8 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 [127]. We first describe the experimentation setup. Then, we examine the effect of gossip parameters on the quality of recommendations, and performance of the system for Rand and Semt. Finally, we evaluate each gossip protocol and its effect on the respective metrics, and the effect of TTL and *query-histories* on query processing.

3.8.1 Experimentation Setup

We use the classical metric of *recall* in IR and RSs to assess the quality of the returned results [135]. Recall represents the system ability to return all relevant documents to a query from the dataset. Thus, in order to measure recall, the relevant document set for each query that has 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 [60] 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 or abstracts from 270 medical journals over a five year period (1987-1991). It was used for the TREC09 Filtering Track [127]. 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 $|R_q|$:

$$Recall = 100 \,.\, \frac{|P_q \cap Rq|}{|Rq|} \qquad (3.12)$$

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:

$$C.coef_u = \frac{\sum_{i=1}^{view\text{-}size_u} loacl\text{-}view_u \cap loacl\text{-}view_{u_i}}{(view\text{-}size_u)(view\text{-}size_u+1)},\ u_i \in loacl\text{-}view_u \qquad (3.13)$$

In order to compute the clustering coefficient of the network, we sum the clustering coefficient of each user u, and divide it over the number of users in the network.

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 [119], 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 as topic t_q of query the maximum component of its V_q, i.e., the maximum w_q^t.

We use a network consisting of 7115 users. Once a user *u* joins the network, it initializes its local-view by selecting randomly a set of users, and adding them into its local-view (as described in Section 4.3). Suppose that the document popularity follows the zipf distribution [24]. Thus, we assume that the number of replicas of a document is proportional to the number of times a document is relevant for a query in *Q*. After distributing randomly the TREC09 documents over the users in the network, these users have 693308 documents, with an average of 97.433 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 at maximum in 10 topics, and relevant at maximum for 5 topics.

P2Prec is built on top of a P2P content sharing system which we generated as an underlying network of 7115 peers (corresponding to users). We use PeerSim [117] 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 distribution, among *u's* topics of interest 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 3.1. Simulation parameters

Parameter	Values
Topics (*T*)	100
TTL	1, 2, 3
Local-view size (*view-size*)	50,70,100
Gossip length (L_{gossip})	5, 10, 20
Gossip period (C_{gossip})	1,30,60 min
Random-view size (R_{size})	40
Semantic-view size (S_{size})	30
Gossip period for random at 2LG (C_{random})	10 min
Gossip period for semantic at 2LG ($C_{semantic}$)	30 min

Table 3.1 summarizes the main simulation parameters that we have used in the experiments. TTL refers to the time-to-live value of a query. C_{gossip} and *view-size* refers to the gossip period and *local-view* size, respectively, when Semt or Rand is used.

While R_{size}, and S_{size} refers to the Random-view and Semantic-view size, respectively, when 2LG is used as described in Section 3.6. Similarly, C_{random} and $C_{semantic}$ refer to the gossip period used for Rand and Semt, respectively, when 2LG is used. L_{gossip} refers to the maximum size of the gossip message transferred during the gossip exchanged.

We performed our experiments under churn, i.e., the network size continuously changes during the run due to users joining or leaving the system. 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.

3.8.2 Trade off: Impact of Gossip

In this experiment we investigate the effect of gossip parameters for Rand, Semt on the quality of recommendations over the respective metrics. The experiments are done by varying the gossip parameters: gossip message size L_{gossip}, gossip period C_{gossip}, and *view-size*. In each experiment, we vary one of the parameters (L_{gossip}, C_{gossip}, *view-size*) and fix the two other parameters. Then, we collect the results for each parameter after each simulation hour. Notice that each experiment is run for 24 hours, which are mapped to simulation time units. We show the results obtained from using Semt, we do not show the results obtained from Rand which illustrate almost similar performance (i.e., almost same gains and same trade-off).

Table 3.2 summarizes the results obtained from the experiments after 24 simulation hours for Semt. The values in Table 3.2 are the average values of the respective metrics that are collected during the simulation time (i.e., we collect the value of each respective metric after each simulation hour, and then we take the average of each respective metric for the 24 simulation hours).

Table 3.2(a) shows the results obtained due to the variation of C_{gossip}. Decreasing C_{gossip} increases the speed of reaching stabilized recall. It takes 1 hour to reach a recall of 48.71% when C_{gossip} is 1 minute and 15 hours to reach the same recall when we increase C_{gossip} to 1 hour. Decreasing C_{gossip} increases the frequent of gossip exchanges between users, and thus users find their similar relevant users rapidly, which in turn increases the speed of reaching higher recall, communication cost, *hit-ratio* and clustering coefficient.

But decreasing C_{gossip} increases the bandwidth consumed by the users, because gossip exchanges are less spaced and thus more frequent. The bandwidth consumed by a user is multiplied by 60 when decreasing C_{gossip} from 1 hour to 1 minute.

Table 3.2(b) shows the results obtained due to the variation of L_{gossip}. Increasing L_{gossip} increases the bandwidth consumed by a user due to gossiping. When L_{gossip} increases from 10 to 50, the bandwidth of a user is increased by a factor of 5. When L_{gossip} increases, more entries are carried out in the gossip message, thus, increasing message size and bandwidth.

But increasing L_{gossip} increases the speed of reaching higher recall, communication cost, *hit-ratio* and clustering coefficient, because exchanging more entries in the gossip message increases the possibility that each user *u* finds new similar relevant users, and then maintains them in its *local-view*.

Table 3.2(c) shows the results obtained due to the variation of *view-size*. Increasing *view-size* increases the number of users at each user *local-view*. This lets each user *u* maintains more number of relevant users in its *local-view* that are similar to *u*, which in turn increases recall, communication cost and *hit-ratio*.

But increasing *view-size* does not have significant impact on the clustering coefficient. Clustering coefficient depends on the overlap between users' *local-views* and *view-size* (see Equation 3.13). The overlap between users' *local-views* is the numerator, and the *view-size* is the denominator of Equation 3.13. Increasing *view-size* lets users maintain more number of similar relevant users in their *local-views*, which in turn may increase the overlap between users' *local-views* (i.e. increasing the numerator of Equation 3.13), and thus increases the cluster coefficient. However, increasing *view-size* increases the denominator of Equation 3.13, and thus decreases the clustering coefficient. Accordingly clustering coefficient value stays almost similar when we varied the *view-size*.

In Rand, we observe that the variation of C_{gossip} and L_{gossip} does not have significant impact over the recall, communication cost, *hit-ratio* and clustering coefficient metrics. That is, due to the random process used in updating *local-views*, and selecting the entries of gossip messages (cf. Section 3.4.3).

For the rest of the simulation, we use 30 minutes for C_{gossip} (simulation time units), 20 for L_{gossip}, and 70 for *view-size* when Rand or Semt is used, because this setting provides good quality of recommendations with acceptable network traffic.

Table 3.2. Impact of gossip parameters

Metric	C_{gossip}=1min	C_{gossip}=30 min	C_{gossip}=1 hour
Recall	50.04	48.71	41.23
Communication cost	21.23	17.56	11.43
Hit-ratio	0.832	0.776	0.697
Background traffic (bps)	122.23	3.976	2.132
Clustering coefficient	0.362	0.358	0.284

(a) Varying C_{gossip} with (L_{gossip} = 20; *view-size* = 70)

Metric	L_{gossip}=10	L_{gossip}=20	L_{gossip}=50
Avg. Recall	45.63	48.71	49.8
Communication cost	14.47	17.56	19.23
Hit-ratio	0.761	0.776	0.795
Background traffic (bps)	2.14	3.976	11.08
Clustering coefficient	0.338	0.358	0.361

(b) Varying L_{gossip} with (C_{gossip} = 30 min; *view-size* = 70)

Metric	*view-size* = 30	*view-size* =70	*view-size* =100
Recall	37.22	48.71	55.67
Communication cost	9.87	17.56	31.78
Hit-ratio	0.763	0.776	0.792
Background traffic (bps)	3.83	3.976	4.07
Clustering coefficient	0.351	0.358	0.347

(c) Varying *view-size* with (L_{gossip} = 20; C_{gossip} = 30 min)

3.8.3 Trade off: Impact of Semt, Rand, and 2LG

In this experiment, 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 protocol (Rand, Semt or 2LG). 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' *local-views*.

Table 3.2 summarizes the results obtained from the experiments after 24 simulation hours for Semt. The values in Table 3.2 are the average values of the respective metrics that are collected during the simulation time (i.e., we collect the value of each respective metric after each simulation hour, and then we take the average of each respective metric for the 24 simulation hours).

Table 3.3 shows the results obtained from the experiments after 24 simulation hours. The values in Table 3.3 are the average values of the respective metric as described in Section 3.8.2. We showed that the background traffic is affected by gossip period (C_{gossip}) and gossip length (L_{gossip}) (see section 3.8.2). We observe that increasing either C_{gossip} or L_{gossip} increases background traffic while decreasing either C_{gossip} or L_{gossip} decreases it. Thus, Rand and Semt are used with the same gossip parameters (C_{gossip} = 30 minutes, and L_{gossip} = 20), so they consume almost the same bandwidth (4 bps). 2LG consumes more bandwidth because four exchange messages are applied each 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 (13.979 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_3, then most probably u_1 and u_3 are similar, and thus produce a clique. In 2LG, the clustering coefficient is moderate between it uses both Rand and Semt,the first favoring randomness while the other favors cliques. Therefore, the clustering coefficient is higher than in Rand but lower than in Semt.

Table 3.3. Impact of Rand, Semt, and 2LG

Metric	Rand	Semt	2LG
Recall	30.7	48.71	42.23
Communication cost	5.04	17.56	10.89
Max. Hit-ratio	0.515	0.776	0.795
Background traffic (bps)	3.484	3.976	13.979
Clustering coefficient	0.073	0.358	0.133

In Figure 3.8, we show the variation of recall, communication cost, and *hit-ratio* versus time for the three algorithms. Figure 3.8(a) shows that the recall keeps increasing at the beginning, and almost stabilizes after 10 hours. At the beginning, the network size is small and many relevant users are not alive. Thus, many irrelevant documents are returned, which reduces recall. Semt increases recall by a factor of 1.6 in comparison with Rand and by a factor of 1.12 in comparison with 2LG. This is because in Semt, a user *u* has in its *local-view* a high number of relevant users that are similar to *u*'s queries. Thus, when *u* submits a query *q*, *q* reaches more relevant users, and thus more relevant documents are returned.

Figure 3.8(b) shows the communication cost of queries for the three algorithms. We set TTL to 1 so that communication cost represents the number of relevant 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 relevant users that are similar to *u*'s demands, and thus, each query is sent to many neighbors. In Rand, the communication cost is low because each *u* has few relevant users to which queries could be sent to. 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 3.8(c) shows the *hit-ratio* for the three algorithms. The maximum *hit-ratio* that has been obtained by Rand is low (0.515). Under Rand, each user *u* has few relevant users that are similar to *u*'s queries. Thus, when *u* submits a query *q*, there is a high probability that *u* does not find a relevant user in its *local-view* that can serve *q*. In Semt and 2LG, the *hit-ratio* is high because *u*'s *local-view* includes many relevant users that are similar to *u*'s demands. Thus, when *u* submits a query *q*, *u* finds many relevant users in its *local-view* that can serve its query *q*.

In Figure 3.8, the significant jump in the beginning of the results is because we start from time zero, no queries are issued and thus no result is gathered. We start the experiments with a small network size, so the number of involved users is small. Therefore, the average value of the metrics starts large, because we divide the gathered value of metrics over a small number of users. As the experiments proceed and more users join the networks, the number of users involved increases and the average value of the metrics stabilizes.

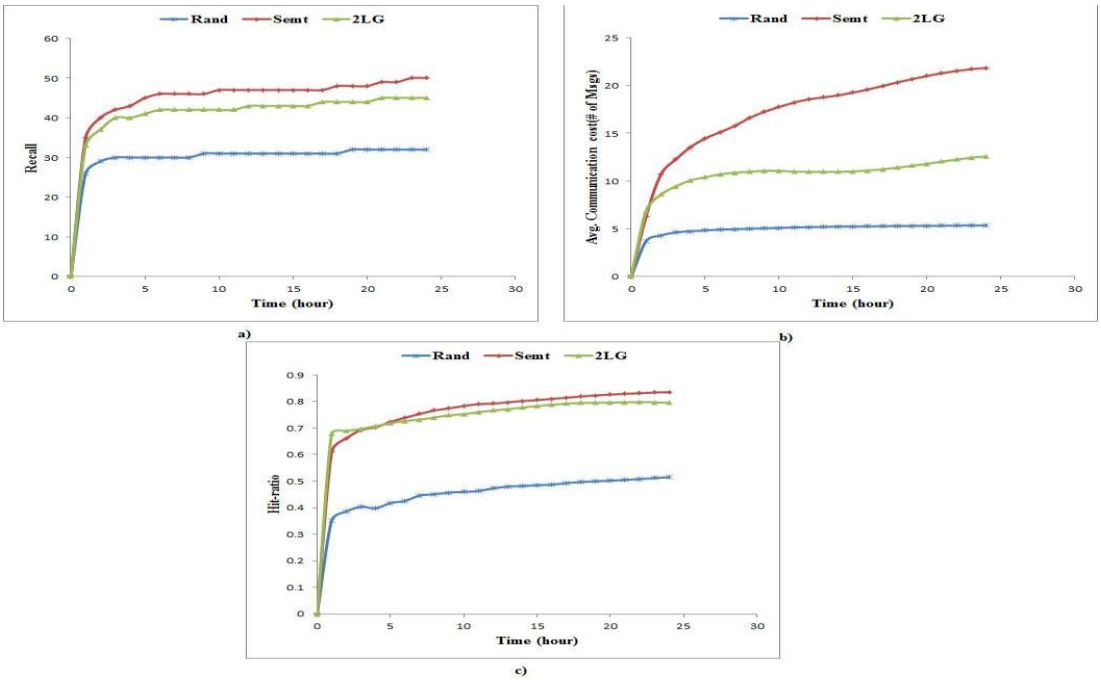


Figure 3.8. The variation of recall, communication cost, and *hit-ratio* versus time

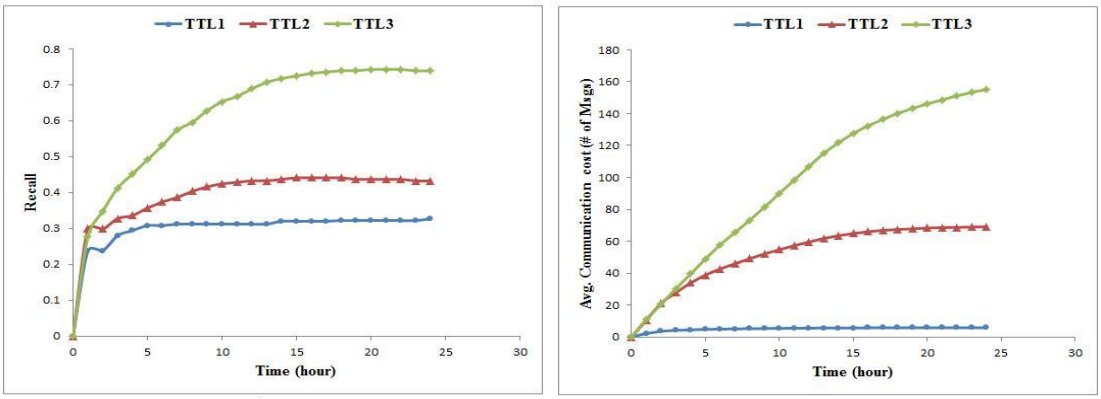
3.8.4 Effect of TTL

We investigate the effect of varying TTL on the quality of recommendations over recall and communication cost metrics. In each experiment, we run one gossip protocol and vary TTL from 1 to 3. Then, we collect the results for each algorithm under each TTL after 24 simulation hours.

Notice that clustering coefficient depends on the overlap between users' *local-views*, and *hit-ratio* depends on the *local-view* of the query's initiator. Accordingly, redirecting query to neighbors of neighbors (i.e., the query's TTL variation) does not have impact on the clustering coefficient and *hit-ratio* metrics.

The TTL variation has significant impact on recall and communication cost especially when Rand is used. When increasing TTL, more relevant users are visited, thus increasing the communication cost and the number of returned documents, which in turn increases recall.

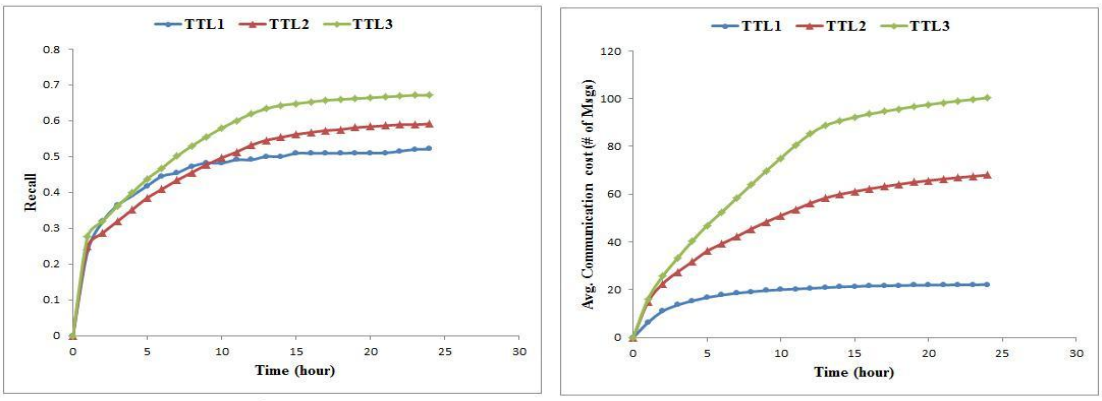
In Figure 3.9 we show the variation of recall and communication cost versus time for the three TTLs when Rand is used. Figure 3.9(a) shows that in Rand the communication cost is multiplied by 26.5 when TTL increases from 1 to 3 (141 relevant users are visited), while maximum recall is increased from 31% to 73.04% (See Figure 3.9(b)).



a) b)

Figure 3.9. The effect of TTL in Rand over recall and communication cost

Figure 3.10 shows the variation of recall and communication cost versus time for the three TTLs when Semt is used. In Semt, maximum recall is increased from 50.1% to 68.5%, when TTL increases from 1 to 3 (see Figure 3.10(a)), while communication cost is multiplied by 4.62 (100 relevant users are visited) (see Figure 3.10(b)). Varying TTL does not have significant impact on Semt, due to the fact that the users' *local-views* have high overlap. Thus, when a user *u* submits a query *q* to a user *v*, *v* does not have many relevant users in its *local-view* that have not received *q* before, because the overlap between *u*'s *local-view* and *v*'s *local-view* is high.



a) b)

Figure 3.10. The effect of TTL in Semt over recall and communication cost

Figure 3.11 shows the variation of recall and communication cost versus time for the three TTLs when 2LG is used. The TTL variation has moderate impact on 2LG. Hence, maximum recall is increased from 43.1% to 82.4% when TTL increases from 1 to 3 (see Figure 3.11(a)), while communication cost is multiplied by 19 (234 relevant users are visited) (see Figure 3.11(b)). Remember 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 relevant users in its semantic and random views that have not received *q* yet.

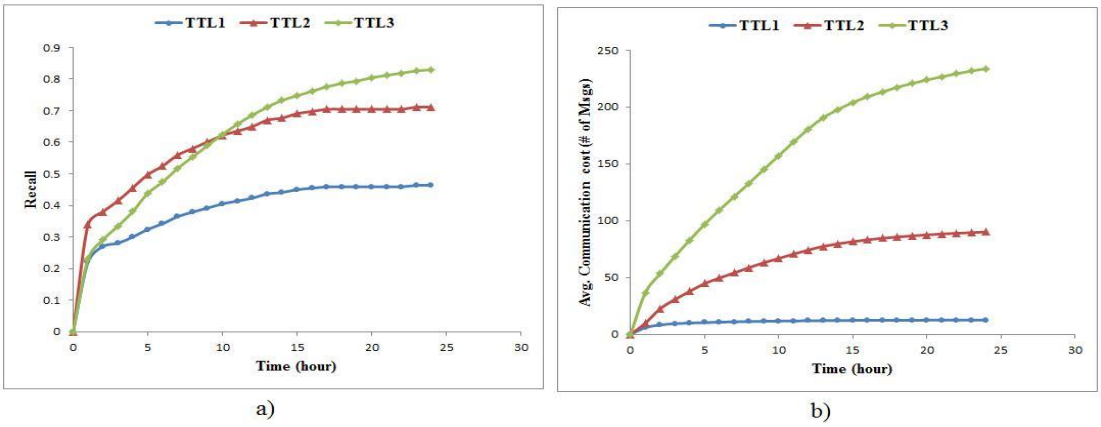


Figure 3.11. The effect of TTL in 2LG over recall and communication cost

3.8.5 Effect of Using Query-histories

In this experiment, we study the effect of using *query-histories* to support failed queries. Figure 3.12 shows the effect of using *query-histories* in the 2LG algorithm with TTL=1 on recall, communication cost and *hit-ratio*. In the fact, the use of *query-histories* increases the *hit-ratio* to 96.9%. That is, each time a user *u* submits a query *q*, there is a high probability to find a relevant 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 relevant users that are most similar to itself. The queries that have been requested by user *u* are most probably similar to queries 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.

Using *query-histories* increases recall slightly, because more users are visited and thus, more documents are returned. But it also increases communication cost, because more relevant users are visited.

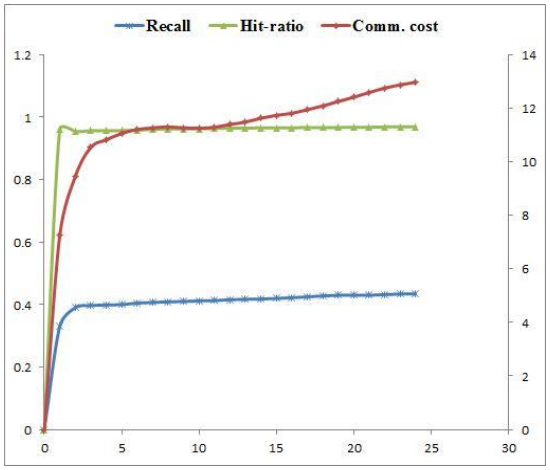


Figure 3.12. The effect of *query-histories* on recall, communication cost and *hit-ratio*

3.9 Conclusion

In this chapter, we proposed P2Prec, a recommendation system for large-scale data sharing that leverages collaborative- and content-based filtering recommendation approaches. P2Prec is useful to recommend to a user documents that are highly related to a specific topic from relevant users in that topic. Each user in the system is automatically assigned topics of interest, based on a combination of topic extraction from its documents (the documents it shares) and ratings. To extract and classify the hidden topics available in the documents, we use the LDA technique. P2Prec is built on top of an unstructured overlay for the sake of scalability and decentralization, and uses gossip protocols to disseminate relevant users and their topics. P2Prec uses two new semantic-based gossip protocols (Semt and 2LG) to let each user aggregate similar relevant users and insure randomness in its view.

In our experimental evaluation, using the TREC09 dataset, we showed that using Semt increases recall and *hit-ratio*. This is because each user maintains in its *local-view* a high number of relevant users that can serve its demands. Using Rand decreases the overlap between users' *local-views* and thus, increases randomness. Using 2LG exploits the advantages of Rand and Semt. It increases recall and *hit-ratio* by a factor of 1.4 and 1.6, respectively, compared with Rand, and reduces the overlap between users' *local-views* by a factor of 2.è compared with Semt.

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

The natural next step (see next Chapter) is to focus on designing a decentralized RS based on explicit personalization (friendship network) and users' topics of interests over a distributed graph, in order to increases the quality and confidence of results, and alleviate the system from cold start problem.

Chapter 4 F2Frec: Leveraging Social- and Content-based Recommendation in P2P Systems

Abstract. In this chapter, we exploit the social relationships between users as a parameter for recommendation, in order to increase the trust and confidence of recommendation. For this, we propose F2Frec that leverages content- and social-based filtering recommendation approaches, in order to construct and maintain a P2P and friend-to-friend network, and to facilitate recommendations. Furthermore, we propose new metrics based on users' relevant topics of interest, such as usefulness, and similarity (among users and their respective friend network), necessary to enable friendship establishment and to select recommendations. In our experimental evaluation, using the Wiki vote social network and TREC09 dataset, we show that F2Frec has the ability to get reasonable recall and precision with acceptable network traffic. Moreover, we show that ranking recommendations put the relevant documents in the top positions of the rank list.

4.1 Introduction

Quality and confidence of recommendations is one of the main requirements that should be taken into account when designing a P2P-RS. In Chapter 3, we described P2Prec a fully decentralized P2P-RS that leverages collaborative- and content-based filtering recommendation approaches, in order to alleviate the scalability problem faced by the state-of-the-art solutions (see Section 2.6). In P2Prec, users' *local-views* are used as a directory to serve their queries, and to generate recommendations. Yet, the users that are maintained at a user *u*'s *local-view* are anonymous to *u*, because they are selected either randomly (as in Rand), or by using the semantic similarity between users' topics of interest (as in Semt). In other words, there is no social relationship such as friendship, trusts, etc. between *u* and the users maintained in its *local-view*, which may deteriorate the quality and confidence of the returned recommendations.

Then, P2Prec should be extended to provide high quality, trustable and confidence recommendations taking into account the nice properties provided by P2Prec such as self-organizing, reliability, scalability, and the high performance. For this, we

proposed F2Frec, which leverages users' topics of interest and social data, in order to maintain P2P social network and generate recommendations.

Sinha et al. [142] 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 recommendation systems.

In existing P2P-RSs, friendship links are extracted from users' behaviors [80][9] or are established based on explicit trust declaration [100]. Siham et al. [9] proposed a generic framework that can be used to extract relations between users based on their tags, items, personal information, etc. in order to extract and extend user's communities, friends etc. 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.

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* as described in Section 3.3.2. 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 [152]. 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 random gossip protocols 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.

Different from P2Prec, where *local-views* are used to serve queries, in F2Frec, a query is forwarded to friends (of friends) users. That is, 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.

The major contributions of this chapter (also in [41] as a preliminary version) can be summarized as follows.

- 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 and Wiki vote social network [165].

The rest of this chapter is organized as follows. Section 4.2 provides a general overview of F2Frec and the problem definition. Section 4.3 introduces F2Frec basic concepts, and presents our social metrics and how we manage friendship establishment. Section 4.4 describes our solution for retrieving recommendations over F2Rrec given a keyword query. Section 4.5 explains how we manage the dynamicity of users' topics of interest. Section 4.6 gives our experimental validation. Section 4.7 concludes.

4.2 General Overview of F2Frec and Problem Definition

F2Frec is a social version of P2Prec that facilitates the construction and maintenance of P2P social network, and exploits social metrics to provide recommendations. In this section, we first give a general overview of F2Frec system. Then we define the problems that F2Frec addresses.

4.2.1 General Overview of F2Frec

F2Frec 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.

To manage topics of interests and relevant users, we reuse the concepts defined in Section 3.3. In short, we use LDA to automatically extract the topics in the system (the training at a global level), which in turn are used to extract users' relevant topics of interest (inference at the local level).

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 [100] or implicitly [87].

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 keywords, 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:

$$recommendation_q = rank(rec_q^{v1}(doc_1),..., rec_q^{vi}(doc_j)) \quad (4.1)$$

Different recommendations may be given for the replicas of a document *doc*. The $recommendation_q$ is ordered based on a ranking function, that ranks each $rec_q^v(doc)$ according to its relevance with q, its popularity, and the distance and trust between the q's initiator and responder v. The trust value between a query's initiator u and a responder v, denoted by $trust_q(u,v)$, is computed during query processing by multiplying the trust values among directs friends along the query path between u and v. More details on query processing, trust computation and recommendations ranking are given in Section 4.4.

4.2.2 Problem Definition

Given the above recommendation model and keyword query q submitted by u, the problems we tackle are:

- The definition of new metrics to enable friendship establishment taking into account users topics of interest and relevant topics.
- The design of a P2P algorithm useful for friendship establishment by taking into account the defined metrics.
- The definition of a criteria to choose the appropriate direct or indirect friends to provide recommendations based on $friends(u)=\{f_1, f_2,...f_n\}$ in a top-k approach.
- Proposal of a tunable, parameterized ranking metric to choose the best recommendations provided by the appropriate friends

4.3 Friend to Friend Recommendation

The goal is to let each user explicitly establish friendship with useful users, so that it can exploit them for recommendation. First, we introduce the basic concepts (FOAF files and random gossip) used in F2Frec. Next, we present the similarity metrics we propose. Finally, we present the data structures and algorithms for friendship establishment.

4.3.1 FOAF File under F2Frec

FOAF 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 [95].

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.

We have adapted the FOAF files to F2Frec and extended the FOAF syntax to also describe users' relevant topics of interest, and the trust between friends. Figure 4.1 shows the FOAF file adapted to F2Frec. 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 relevant topics of interest. It shows that she is interested in topics $t_1 \in T$ and $t_2 \in T$, and relevant in topic t_2. Given that t_1 and t_2 have been extracted from the documents she maintains by using the global and local inference 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. Peter is interested in t_1, t_2 and t_3, but he is relevant in topic t_1. In addition, Jean has trusted Peter by a value of 0.8.

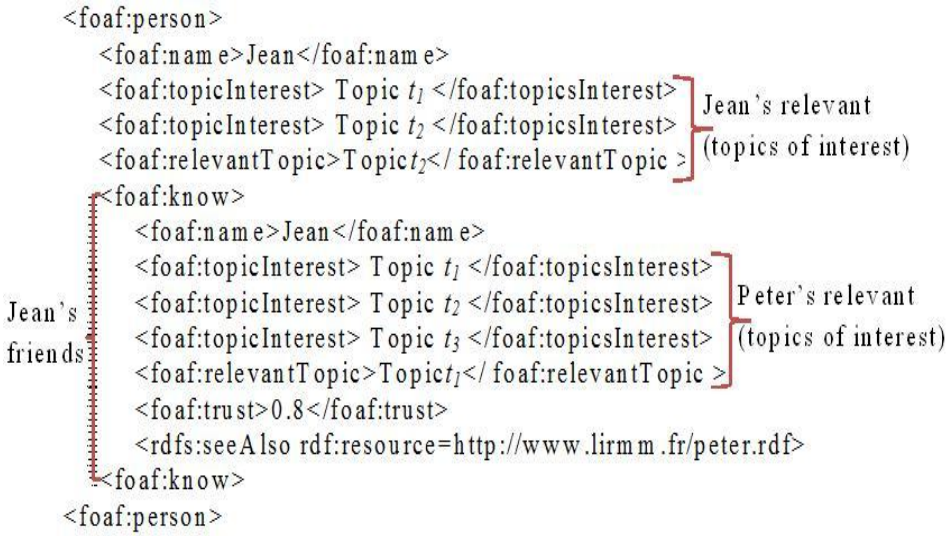


Figure 4.1. An example of a FOAF file in F2Frec

4.3.2 Random Gossip under F2Frec

The information about relevant users is disseminated using Rand in order to guide users in establishing new useful friends, and updating their FOAF files accordingly. Recall that each user *u* maintains a *local-view*, which contains a fix number of entries, where each entry refers to a user *v*, and contains *v*'s gossip information such as: 1) *v*'s IP address. 2) *v*'s topics of interest T_v, each topic $t \in T_v$ being associated with a Boolean field that indicates whether *v* is relevant in that topic. F2Frec extends *local-view*'s entry to include additional gossip information that is necessary for computing the metrics such as: 3) *v*'s friends built using a Bloom filter [20]. 4) *v*'s friendship network size i.e., |*friend(v)*| helpful to the metric. 5) The timestamp of the entry: a numerical field represents the version of the entry, where greater timestamp means fresher entry.

In F2Frec, we adopt Rand (described in Section 3.4.3) with a modification to its update process only, to take advantage of the entries' timestamp. That is, when a user *u* receives a gossip message, it updates its *local-view* based on the gossip message received taken into account the entries' timestamp. 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, and discards the duplicates: if 2 entries related to the same user, only the instance with the largest timestamp value is kept. 2) Using the buffer, *u* selects *view-size* entries randomly and updates its *local-view*.

4.3.3 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:

$$distance_{fri}(u,v) = \frac{2|friend(u) \cap friend(v)|}{|friend(u)|+|friend(v)|} \quad (4.2)$$

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:

$$distance_{intr}(u,v) = \frac{2|T_u \cap T_v|}{|T_u|+|T_v|} \quad (4.3)$$

We use the $distance_{intr}(u,v)$ metric to rank recommendations, more details are given in see Section 4.4.3. 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)*:

$$useful(u,v) = \frac{2|T_u \cap T_v^r|}{|T_u|+|T_v^r|} \quad (4.4)$$

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.

$$sim(u,v) = \alpha * useful(u,v) + (1-\alpha) * distance_{fri}(u,v) \quad (4.5)$$

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*:

$$useful(q,v) = \frac{2|T_q \cap T_v^r|}{|T_q|+|T_v^r|} \quad (4.6)$$

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

4.3.4 Friendship Establishment

Algorithm 4.1 shows how each user *u* exploits its gossip *local-view* to establish friendship. For each gossip cycle, *u* goes through each user $v \in local\text{-}view_u$, and evaluates whether *v* may be suggested for friendship as follows. User *u* computes the simi-

larity distance *sim(u,v)* as described in Section 4.3.3 (lines 3-5). User *v* is suggested to *u* for friendship if similarity *sim(u,v)* exceeds system-defined threshold, denoted by τ, (lines 6 and 7), which is:

$$sim(u,v) \geq \tau \tag{4.7}$$

Notice that the suggestion may include 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 (lines 8 and 9). Then, *u* adds *v* to a *waitList* list (line 10), waiting for friendship confirmation.

Afterwards, user *u* receives a reply message, denoted by msg_{rep}, from each user $v \in waitList$ (line 15). 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 (line 16-19). Notice that the *trust(u,v)* is assigned explicitly by *u* [100].

Algorithm 4.1- How a user *u* establishes friendship

Input: $local\text{-}view_u$
Output: updated $FOAF_u$

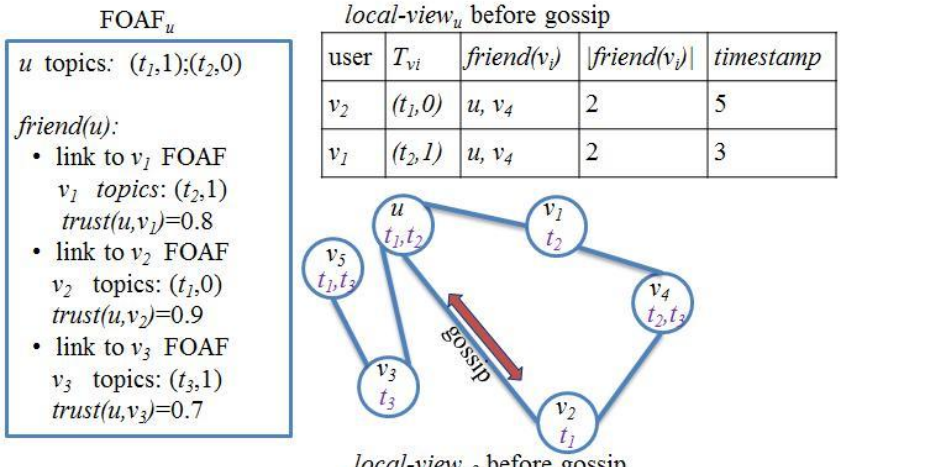
Forever do
  For user $v \in local\text{-}view_u$ && $v \notin friend(u)$ **do**
    u **computes** *useful(u,v)*
    u **computes** $distant_{fri}(u,v)$
    $sim(u,v) = \alpha * useful(u,v) (1-\alpha) * distant_{fri}(u,v)$
    IF $sim(u,v) \geq \tau$ **then**
      suggests *v* to user *u*
        IF user *u* **accepts** the suggestion **then**
          u **sends** msg_{req} to user *v*
          u **adds** *v* to *waitList*
        End If
    End If
  End For
  For each user $v \in waitList$ **do**
    u **receives** msg_{rep} from user *v*
    IF $msg_{rep} = accept$ **then**
      u **adds** *v* to its FOAF file as a friend
      u **adds** T_v and T_v^r to its FOAF file
      u **assigns** *trust(u,v)* and **adds** to its FOAF
    End If
  End For

Figure 4.2, shows an example of F2Frec gossip exchange and friend establishment. In the example, the network is composed by 7 users that are interested in three topics. The link between two pair of users represents that they are friends. Figure 4.2 also shows the content of users' FOAF files and *local-views* before and after gossip exchange.

Before gossip exchange (Figure 4.2(a)), u has three friends v_1, v_2, and v_3. Each friend is relevant (interested in a set of topics), and a declared trust. For instance, v_1 is a friend to u, v_1 is relevant and interested in topic t_2, and u has declared trust with v_1 by a level of 0.8. Also, it shows that u maintains in its *local-view* two entries that refer to users v_1 and v_2, and includes their gossip information. For instance, the entry that refers to v_2 indicates that v_2 is interested in t_1, has two friends u and v_4, and the timestamp of entry is 3.

Suppose u has selected v_2 to gossip with, and then a message exchange has been performed between u and v_2 (see Figure 4.2(a)). Afterwards, u updates its *local-view* based on the gossip message it has received from v_2 (see Figure 4.2(b)). Therefore, u's *local-view* contains entries for users v_4 and v_6 as shown in Figure 4.2(b).

After that, u evaluates whether users v_4 and v_6 may be suggested for friendship. First, u measures the similarity between itself and users v_4 and v_6 based on the Equation 4.5. Suppose the value of α in Equation 4.5 is equal to 0.5, then the $sim(u,v_4)$ is equal to 0.65 and $sim(u,v_5)$ is equal to 0.5. Suppose that the value of τ in Equation 4.7 is equal to 0.5, then v_4 is suggested to u, and friendship establishment is lunched between u and v_4 as depicted in Figure 4.2(c). Afterwards, u becomes friend to v_4. Thus u stores v_4 in its FOAF file along with its topics of interest t_2 and t_3, and u has declared trust with v_4 by a level of 0.75 (see Figure 4.2(d)).



local-view$_u$ before gossip

user	T_{vi}	*friend*(v_i)	|*friend*(v_i)|	*timestamp*
v_2	$(t_1,0)$	u, v_4	2	5
v_1	$(t_2,1)$	u, v_4	2	3

local-view$_{v2}$ before gossip

user	T_{vi}	*friend*(v_i)	|*friend*(v_i)|	*timestamp*
v_4	$(t_2,1);(t_3,0)$	v_1, v_2	2	3
v_5	$(t_1,1);(t_3,1)$	v_5	1	4

a) *local-view* before gossiping

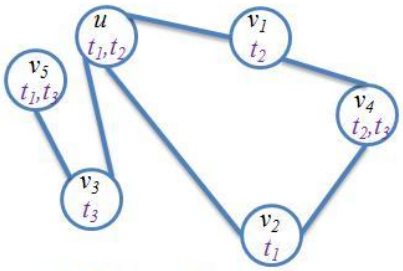
FOAF$_u$

u topics: $(t_1,1);(t_2,0)$

friend(u):
- link to v_1 FOAF
  v_1 *topics*: $(t_2,1)$
  trust(u,v_1)=0.8
- link to v_2 FOAF
  v_2 topics: $(t_1,0)$
  trust(u,v_2)=0.9
- link to v_3 FOAF
  v_3 topics: $(t_3,1)$
  trust(u,v_3)=0.7

local-view$_u$ after gossip

user	T_{vi}	*friend*(v_i)	$\|friend(v_i)\|$	*timestamp*
v_4	$(t_2,1);(t_3,0)$	v_1, v_2	2	3
v_5	$(t_1,1);(t_3,1)$	v_5	1	4



local-view$_{v2}$ after gossip

user	T_{vi}	*friend*(v_i)	$\|friend(v_i)\|$	*timestamp*
v_4	$(t_2,1);(t_3,0)$	v_1, v_2	2	3
u	$(t_1,1);(t_2,0)$	v_1, v_2, v_3	3	6

b) *local-view* after gossiping

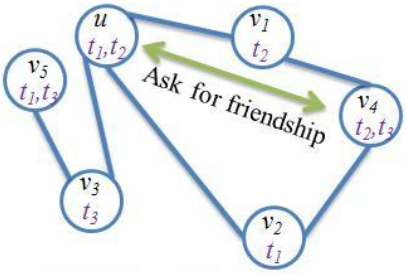
FOAF$_u$

u topics: $(t_1,1);(t_2,0)$

friend(u):
- link to v_1 FOAF
  v_1 *topics*: $(t_2,1)$
  trust(u,v_1)=0.8
- link to v_2 FOAF
  v_2 topics: $(t_1,0)$
  trust(u,v_2)=0.9
- link to v_3 FOAF
  v_3 topics: $(t_3,1)$
  trust(u,v_3)=0.7

local-view$_u$ after gossip

user	T_{vi}	*friend*(v_i)	$\|friend(v_i)\|$	*timestamp*
v_4	$(t_2,1);(t_3,0)$	v_1, v_2	2	3
v_5	$(t_1,1);(t_3,1)$	v_5	1	4



local-view$_{v2}$ after gossip

user	T_{vi}	*friend*(v_i)	$\|friend(v_i)\|$	*timestamp*
v_4	$(t_2,1);(t_3,0)$	v_1, v_2	2	3
u	$(t_1,1);(t_2,0)$	v_1, v_2, v_3	3	6

c) Establishing friendship

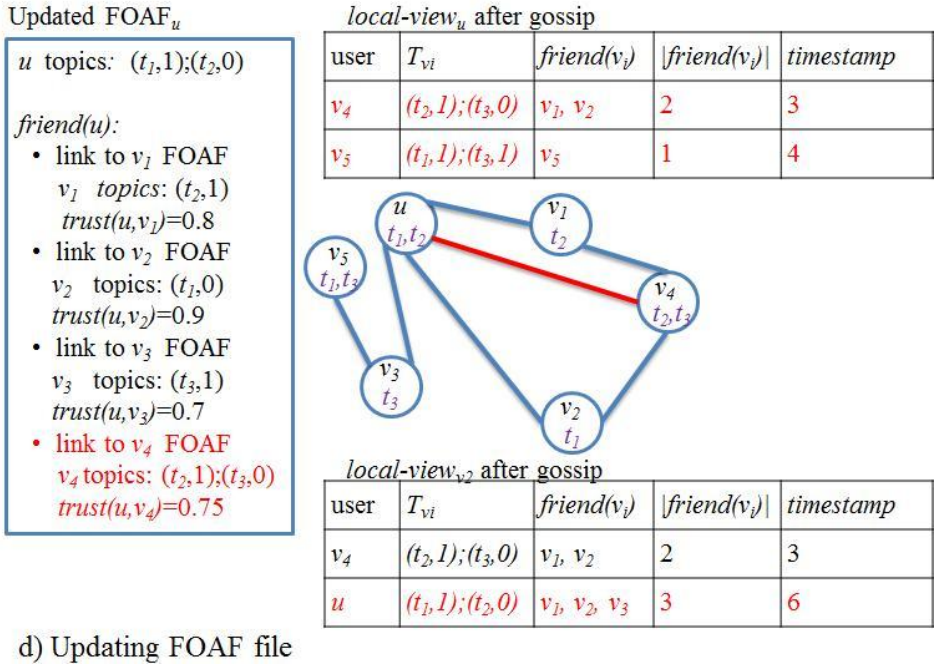


Figure 4.2. Snapshot of F2Frec System

4.4 Query Processing based on FOAF Files

In this section, we describe our query processing algorithm to generate recommendations. Next we explain the criteria we use to compute the trust level between the query's initiator and a responder. Finally, we describe the ranking model we use to order the returned recommendations.

4.4.1 Query Processing

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.

Algorithm 4.2 illustrates the behaviors of query processing. In active behavior, a user u issues a query q and proceeds as follows. First, it computes how much each useful friend $v \in friend(u)$ is useful to q (line 2). 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:

$$rank(v) = trust(u,v)*useful(q,v) \quad (4.8)$$

Once *u* has computed the rank of each useful friend *v*, it adds *rank(v)* to a *RankList* (lines 3 to 5) that contains the useful friends' addresses along with their ranks. Then, it selects the top-k useful friends from the *RankList* with highest rank using **SelectTopk** (), and adds them to *topkList* (line 8). 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 (lines 9 to 14). 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 (lines 15 to 17) (see Section 4.4.3).

In the passive behavior, 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 (lines 3 and 4). 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 $V_{doc} = [w_{doc}^{t1},\ldots,w_{doc}^{tk}]$ and the query topic vector $V_q = [w_q^{t1},\ldots,w_q^{tk}]$, which is:

$$sim(doc,q) = \frac{\sum_{i=1}^{d} w_q^{t_i} * w_{doc}^{t_i}}{\sqrt[2]{\sum_{i=1}^{d} w_q^{t_i} * w_q^{t_i} * \sum_{i=1}^{d} w_{doc}^{t_i} * w_{doc}^{t_i}}} \quad (4.9)$$

Second, *v* returns to the query's initiator *u* the recommendations for the documents whose similarity exceeds a given (system-defined) threshold (lines 5 and 6).

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 (lines 9 to17). Then, *v* computes the trust value $trust_q(u,x)$ for each useful friend $x \in topkList$ based on Equation 4.9 (line 19) (presented in Section 4.4.2). Then *v* attaches $trust_q(u,x)$ to *q*, and forwards *q* to *x* after reducing TTL by one (lines 20 to 22).

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.

Algorithm 4.2- Query Processing

//Active behavior: **Route-Query**(q, $FOAF_u$)
Input: query q ($word_i$, TTL, V_q, T_q,u); $FOAF_u$
Output: submit q to potential friends; recommendations

1 **For** each useful friend $v \in friend(u)$ **do**
2 user u **computes** $useful(q,v)$
3 **If** $useful(q,v)$ >0 **then**
4 $rank(v) = trust(u,v)*useful(q,v)$
5 user u **adds** <$rank(v)$,v> to $RankList$
6 **End if**
7 **End For**
8 $topkList$ = **selectTopk**($RankList$)
9 **For** each useful friend $v \in topkList$ **do**
10 $trust_q(u,v) = trust(u,v)$
11 u **attaches** $trust_q(u,v)$ to q
12 $q.TTL = q.TTL-1$
13 u **send** q to v
14 **End For**
15 **If** user u **Receives** $rec_q^{v1}(doc_1),\ldots, rec_q^{vi}(doc_j)$ **then**
16 u **ranks** ($rec_q^{v1}(doc_1),\ldots, rec_q^{vi}(doc_j)$)
17 **End If**

//Passive behavior: **Process-query**(q, D_u, $FOAF_u$)
Input: q ($word_i$, TTL, V_q, T_q, $trust_q(x,u)$); D_u; $FOAF_u$
Output: recommendations for query q; u send q to top-k useful friends

1 **Forever do**
2 **Receive** query q initiated by x
3 **For** each $doc \in D_u$ **do**
4 $Sim(q,doc)$ = **CosineSimilarity**(V_q,V_{doc})
5 **If** $Sim(q,doc)$ greater than $threshold$ **then**
6 **recommend** doc to q's initiator
7 **End If**
8 **End For**
9 **If** $q.TTL$ not equal to zero **then**
10 **For** each useful friend $v \in friend(u)$ **do**
11 user u **computes** $useful(q,v)$
12 **If** $useful(q,v)$ >0 **then**
13 $rank(v) = trust(u,v)*useful(q,v)$
14 user u **adds** <$rank(v)$,v> to $RankList$
15 **End if**
16 **End For**
17 $topkList$ = **selectTopk**($RankList$)
18 **For** each useful friend $v \in topkList$ **do**
19 $trust_q(x,v) = trust_q(x,u)*trust(u,v)$
20 u attaches $trust_q(x,v)$ to q
21 $q.TTL = q.TTL-1$
22 u **send** q to v
23 **End For**
24 **End If**

4.4.2 Trust Computation

We compute the trust value between a query's initiator and a responder during the query processing (line 19 in the passive behavior of Algorithm 4.2). When an initiator *u* sends a query *q* to a direct useful friend *v*, it attaches with *q* the trust level $trust_q(u,v)$ which is equal to the *trust(u,v)*. When a user *v* receives a copy of *q*, and the query's TTL is not yet zero, it redirects *q* to a direct useful friend *x* as follows. First it computes the trust value between *u* and *x* $trust_q(u,x)$, by multiplying the trust value $trust_q(u,v)$ attached to *q* that *v* has received, with the trust value *trust(v,x)*. Then *v* attaches $trust_q(u,x)$ to *q*, and forwards *q* to *x*.

More in details, the path of a query *q* between an initiator *u* and a responder *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:

$$trust_q(u,v) = \prod_{v_i,v_j \in path_q(u,v)} trust(v_i,v_j) \quad (4.10)$$

With such trust method, we do not need an extra data and information to propagate and aggregate the trust network, thus minimizing the complexity and network traffic of the system. This trust method does not represent the most optimal trust value between the initiator *u* and the responder *v*. However, it gives a good approximation, because the users involved in the $path_q(u,v)$ are the most trustful friends (of friends).

4.4.3 Ranking Recommendations

Recall that the result of a query *q* submitted by a user *u* is $recommendation_q = rank(rec_q^{v1}(doc_1),\ldots, rec_q^{vi}(doc_j))$, where $rec_q^v(doc)$ is the recommendation that has been given for a document *doc* from a responder *v*. We rank $rec_q^v(doc)$ based on the semantic similarity between *q* and *doc*, the popularity of *doc*, and the distance and trust between *u* and the responders of *doc*. Accordingly, $rec_q^v(doc)$ that has been received from responder *v* includes *sim*(*doc*,*q*), *v*'s topics of interest T_v and the $trust_q(u,v)$. The rank of a $rec_q^v(doc)$ (line 15 in the active behavior of Algorithm 4.2), denoted by $rank(rec_q^v(doc))$, is defined as:

$$rank(rec_q^v(doc)) = a * sim(doc,q) + b * \frac{\sum_{v \in R} distance_{intr}(u,v) * trust_q(u,v)}{|R|} + c * pop(doc) \quad (4.11)$$

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

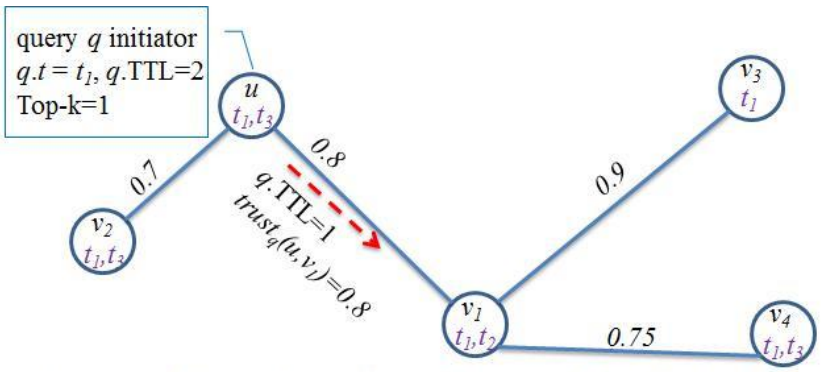
a, *b* and c. Upon receiving the recommended documents, user *u* can download a copy of a document, rate and include it in its document set D_u.

Figure 4.3 shows an example of query processing in F2Frec. The network is composed of a user *u* and its friends (of friends) along with their relevant topics and trust. In the example, suppose that user *u* initiates a query *q* for topic t_1 with TTL=2 and *k*=1 (see Figure 4.3(a)). User *u* ranks its friends based on trust, and their usefulness to *q*. Based on *k* and friends' ranks, *u* forwards *q* to v_1, because v_1 has the highest rank as shown in Figure 4.3(a). But before forwarding, *u* attaches $trust_q(u,v_1)$ to *q* and reduces TTL by one. Notice that $trust_q(u,v_1)$ is equal to $trust(u,v_1)$, which is equal to 0.8, because *u* and v_1 are direct friends.

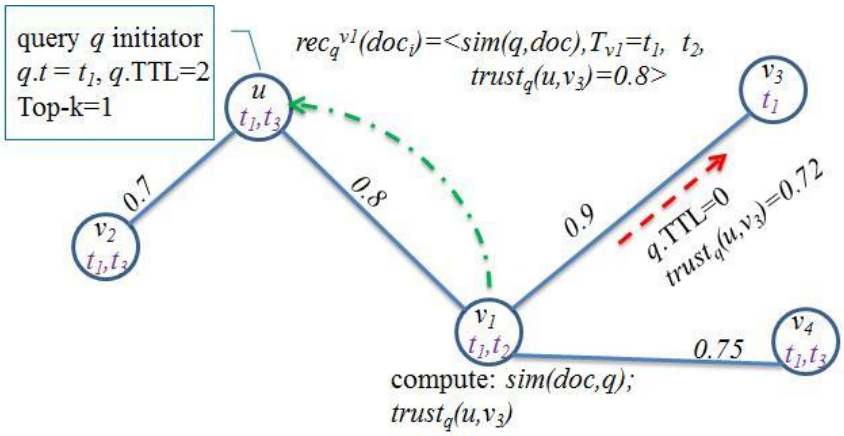
Figure 4.3(b) shows the behavior of v_1 when it receives *q*, v_1 computes the similarity between *q* and its documents $sim(doc,q)$ where $doc \in D_{v1}$. Then it returns to *u* the recommendations $rec_q^{v1}(doc)$ for those documents whose similarity exceeds a given threshold. In addition, $rec_q^{v1}(doc)$ includes v_1's topics of interest T_{v1}, and the trust value between *u* and v_1 that it is attached in the received *q* i.e., $trust_q(u,v_1) = 0.8$.

Since TTL is not equal to zero, v_1 redirects *q* to its top-k trust and useful friends as follows. First, it ranks its useful friends based on trust and usefulness, and forwards *q* to the top rank one i.e., it forwards *q* to v_3, because v_3 has the highest rank. Before forwarding *q* to v_3, v_1 computes the trust between *u* and v_3 $trust_q(u,v_3)$, by multiplying the $trust(v_1,v_3)$ with the trust attached to *q* $trust_q(u,v_1)$ i.e., $trust_q(u,v_3) = 0.8*0.9 = 0.72$. Then, v_1 attaches $trust_q(u,v_3)$ to *q*, and reduces TTL by one.

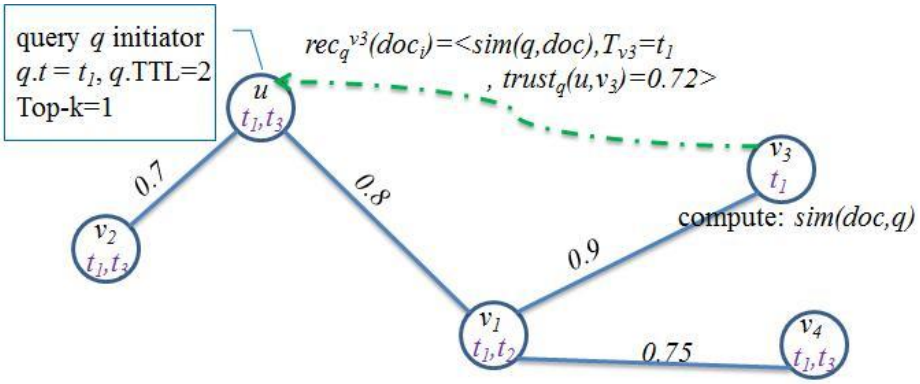
Figure 4.3(c) shows that when v_3 receives *q*, it computes $sim(doc,q)$ for $doc \in D_{v3}$ and returns the recommendations $rec_q^{v3}(doc)$ to *u*. User v_1 does not forward *q* because its TTL has reached zero. Finally, in Figure 4.3(d) we show that when the user *u* receives $rec_q^{v}(doc)$ from useful friends (of friends) v_1 and v_3, it ranks $rec_q^{v}(doc)$ based on their popularity, semantic similarity, distance and trust between *u*, v_1 and v_3 to provide $recommendations_q$.



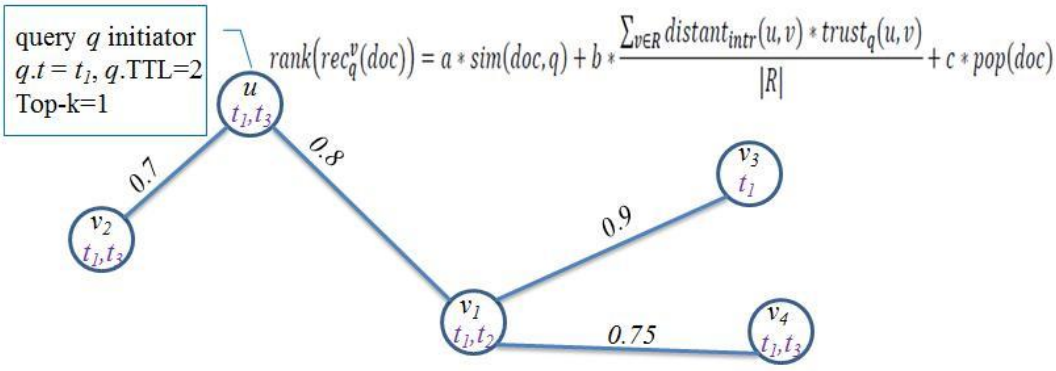
a) Initiator *u* forwards *q* to the top-k trust and useful friends



b) Responder v_1 returns recommendations, and redirects q



c) Responder v_1 returns recommendation, and stops redirecting q



d) Initiator u ranks the returned recommendations

Figure 4.3. Query processing, recommendation ranking, trust computing

4.5 Managing the Dynamicity of Users' Relevant Topics of Interest

Users in F2Frec are active and continuously changing their relevant topics of interest. For instance, users are continuously downloading, rating new documents, and removing some old documents. Consequently, users' relevant topics of interest are changed, and thus their entries should be updated accordingly.

Consider a user u that is relevant in a topic t_1, joins F2Frec and starts gossiping with other users. After a while, there will be a set of users $S \subset U$ that carry an entry for u in their *local-views*, where u's entry indicates that u is relevant in topic t_1. Assume that meanwhile u has changed its relevant topic to t_2. But u is not suggested to any user $v \in S$ that is interested in t_2 for friendship establishment. That is because the entry of u that has been carried by v's *local-view* is not useful to v. In addition when u changes its relevant topics of interest, its friends should update their FOAF files to avoid false redirection of queries. In order to avoid the false friendship establishment and redirection of queries, we propose *friend-promotion* that is inspired on *self-promotion* approach [14].

In *friend-promotion*, whenever a user u changes its relevant topics of interest, it creates a new entry for itself, in which it includes its new relevant (topics and interests) and timestamp. Then, u sends to each friend $v \in friend(u)$ an *update message* carrying the new entry.

Once a friend v receives an *update message*, v updates u's relevant topics of interest in its FOAF file. Then, v includes the new entry of u in its *local-view* as follows. User v checks whether it has an entry for u in its *local-view*. If its the case, v replaces the old entry of u with the new one. In case that v does not have an entry for u in its *local-view*, it adds the new entry of u to its *local-view* while $|local\text{-}view_v| < view\text{-}size$. If v's *local-view* size is equal to *view-size*, v selects randomly a user x from its *local-view*, and replaces the entry of x by the new entry of u.

4.6 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 and the Wiki vote social network [165]. We first describe the experimentation setup. Then, we investigate the effect of gossip on the quality of friendship establishment, and we evaluate the effect of friendship establishment on the performance of F2Frec. After that, we investigate the trade-off of the top-k query routing, and ranking recommendation. Finally, we evaluate the effectiveness of *friend-promotion* algorithm in updating users' *local-views*.

4.6.1 Experimentation Setup

To validate our simulation experiments, we use the TREC09 dataset that was previously used and described in Section 3.8.1. Also, in order to assess the quality of recommendations, in addition to *recall* that was used and discussed in Section 3.8.1, we use *precision* metric. Precision represents the system ability to return documents that are mostly relevant to a query from the dataset.

TREC09 includes 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, when a user u issues a query $q \in Q$, u 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 precision. Thus, precision is defined as the percentage of q's relevant documents $doc \in R_q$ occurring in P_q with respect to the size of P_q i.e., $| P_q |$:

$$precision = 100 \cdot \frac{|P_q \cap R_q|}{|P_q|} \tag{4.12}$$

Moreover, we use **communication cost, hit-ratio** and **background traffic** metrics that were previously used and discussed in Section 3.8.1 in order to assess the search efficiency. In addition, we introduce the **average number of friends** metric in order to assess the effectiveness of friendship establishment.

- **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), which is:

$$avgerage\ number\ of\ friends = \frac{\sum_{u \in U} |friend(u)|}{|U|} \tag{4.13}$$

Where $|U|$ is the cardinality of the set of users in the system, and $|friend(u)|$ is the cardinality of the set of friends for each user $u \in U$.

We use the Wiki vote social network [165] to give randomly each user a set of documents from TREC09. Wiki vote considers that two users are 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.

Our evaluation methodology reuses many concepts defined in Section 3.8.1. In short, we used GibbsLDA++ to extract the document topic vectors V_{doc}, and query topic vectors V_q, with $|T|$=100. We consider that each query $q \in Q$ has one topic in most of experiments setting. We consider that each user u is interested at least in one topic and at most in 10 topics. Also, u is relevant at least for one topic and for 5 topics at most. We use PeerSim for simulation, and generate an underlying network of 7115 nodes, which is equal to the number of users in the Wiki vote network. All experiments are performed under churn, and run for 24 simulation hours. Besides, 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 as described in Section 3.8.1.

In addition, each user $u \in U$ gets its initial friends from the Wiki social network, and then u runs the F2Frec algorithm to establish new friends. We let each user u establish new friends after each time it performs a Rand gossip. Hence, a user u adds a user v as a new friend, if $sim(u,v)$ exceeds 0.5 i.e., $\tau = 0.5$.

Table 4.1 lists the main simulation parameters that we have used in the experiments. C_{gossip}, *view-size*, and L_{gossip} refers to the gossip period, *local-view* size, and the maximum size of the gossip message transferred during the gossip exchanged, respectively. α refers to scale parameter that is used in the Equation 4.5. τ is the threshold value that is used in Equation 4.7 in order to decide whether to suggest a user for friendship establishment.

Table 4.1. Simulation parameters

Parameter	Values
Topics (T)	100
TTL	1, 2, 3
Local-view size (*view-size*)	10,50,70
Gossip length (L_{gossip})	5, 10, 20
Gossip period (C_{gossip})	1, 30, 60 min
α	0.0, 0.3, 0.5, 0.7, 1.0
τ	0.5

4.6.2 Impact of Gossip

In this experiment we investigate the effect of gossip on the friendship establishment over background traffic and average number of friend metrics. The experiments are done by varying the gossip parameters, where in each experiment, we vary one of the parameters (L_{gossip}, C_{gossip}, *view-size*) and fix the other two parameters. We use the value 0.5 for the scale parameter α in Equation 4.5.

Table 4.2 lists the maximum results obtained from the experiments after 24 simulation hours, in terms of average number of friends and background traffic. Table 4.2(a) shows that decreasing C_{gossip} from 1 hour to 1 min increases the average number of friends from 90.2 to 781.5. When C_{gossip} is decreased, the gossip exchanges are less spaced and more frequent, which in turn increases the frequent of performing the friendship establishment. Accordingly, more users are suggested for friendship establishment, and thus more friends are added to users' FOAF files. Moreover, increasing

the frequent of gossip exchange increases the possibility of exploring new relevant users, which in turn increases the possibility of establishing new friendship.

However, decreasing C_{gossip} increases the bandwidth consumed by the user significantly. That is due to two factors. 1) Gossip exchanges are less spaced and thus more frequent. 2) Increasing users' friend increases the size of the gossip entries, which increases the bandwidth. Decreasing C_{gossip} from 1 hour to 1 min increases the bandwidth consumed by a user from 7 bps to 5560 bps.

Table 4.2(b) conveys the fact that carrying more entries in L_{gossip} leads to greater average number of friends, as well as greater background traffic. Increasing L_{gossip} from 5 to 20 increases the average number of friends from 95.5 to 223.85, and background traffic from 7.5 bps to 41.3 bps.

From Table 4.2(c), we can see that using very small *view-size* reduces the possibility of establishing new friendship. That is, because using very small *view-size*, a few relevant users are maintained at users' *local-views*, which reduces the number of users suggested for friendship establishment, and thus a few new friends are added to users' FOAF files.

Table 4.2. Impact of gossip on friendship establishment

C_{gossip} (min)	Avg. Background traffic (bps)	Avg. number of friends
1	5560	781.5
30	19	177.5
60	7	90.2

(d) Varying C_{gossip} with (L_{gossip} = 10; *view-size* = 50)

L_{gossip}	Avg. Background traffic (bps)	Avg. number of friends
5	7.5	95.5
10	19	177.5
20	41.3	223.85

(e) Varying Lgossip with (Cgossip = 30 min; view-size = 50)

View-size	Avg. Background traffic (bps)	Avg. number of friends
10	10.5	78.95
50	19	177.5
70	20.8	192.65

(f) Varying *view-size* with (L_{gossip} = 10; C_{gossip} = 30 min)

In Figure 4.4, we show the variation of the average number of friends versus time under the three gossip parameters. From Figure 4.4 we observe that the average number of friend keeps on increasing with time, given that more new relevant users are explored due to gossiping, more users are suggested for friendship establishment, and thus more friends are added to users' FOAF files. Figure 4.4(a) shows that using very large *view-size* does not have significant impact on the average number of friends. In large *view-size*, when gossip exchange occurred, a few relevant users are added to the *local-views* compared to the *view-size*. Taking into account that the previous entries in the *local-views* are already checked for friendship establishment, then a few users are suggested for friendship establishment, and thus a few new friends are added to users' FOAF files.

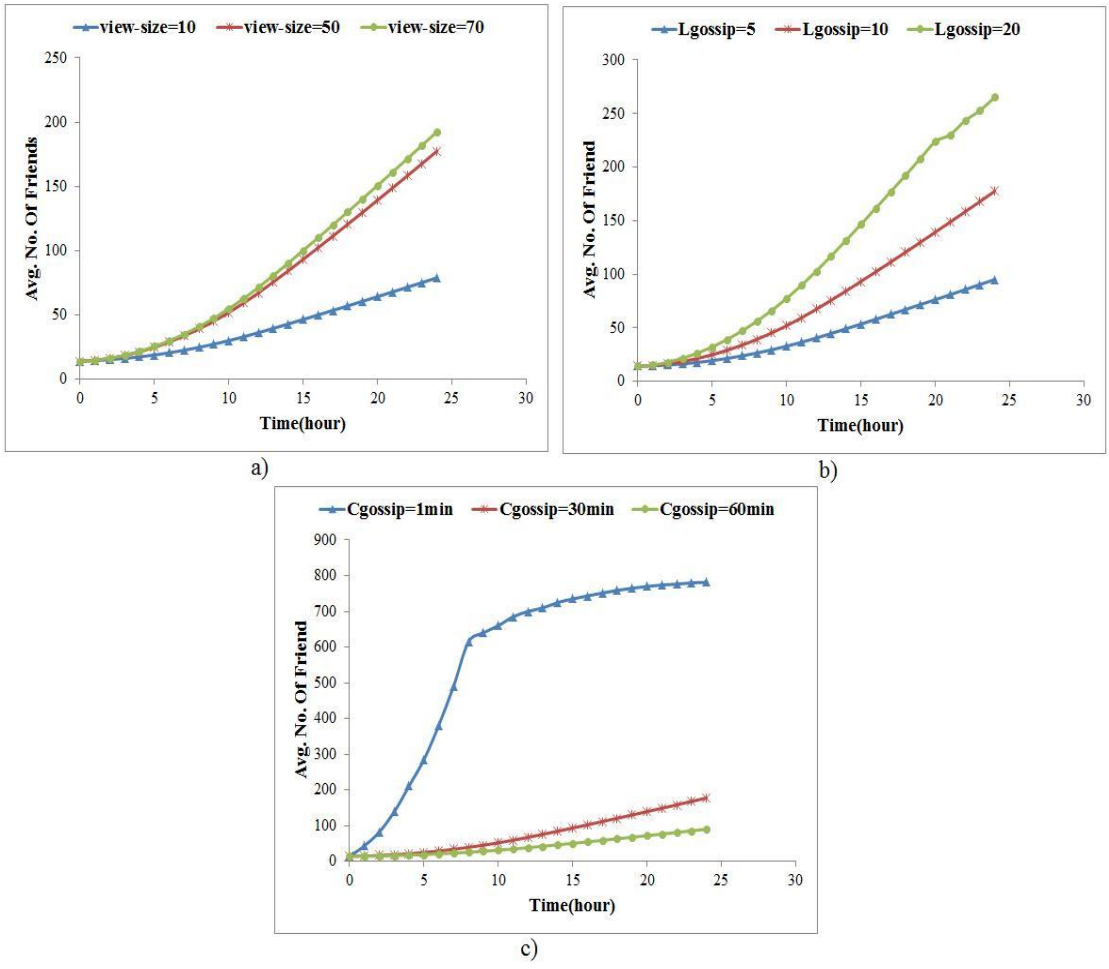


Figure 4.4. The variation of average number of friend versus time

Figure 4.4(b) shows that increasing L_{gossip} increases the possibility of exploring and finding new relevant users at each gossip cycle. Accordingly, the number of users suggested for friendship is increased, which increases the speed of achieving larger average number of friends. For instance, average number of friends reaches 95 after 11 gossip cycles when L_{gossip}=20. Hence, when L_{gossip}=5, it needed 24 gossip cycles to reach average number friend equal to 95.

From Figure 4.4(c) we observe that decreasing the C_{gossip} achieves larger average number of friends very fast. Also we observe that, the average number of friends is stabilized after performing a certain numbers of gossip cycles. The average number of friend stabilizes almost after 9 hour (540 gossip cycles) when 1 min is used for C_{gossip}.

To conclude, we observe that the choice of the 2 gossip parameters (L_{gossip} and C_{gossip}) is a trade-off between the speed of achieving larger average number of friends and the background traffic consumption. Increasing the frequent of gossip exchange, more new relevant users are explored and suggested for friendship, thus larger average number of friends is achieved fast, and more background traffic is consumed. Similarly, the more entries are carried during the gossip exchange, the faster users discover new friends, and more background traffic is consumed.

For the rest of the simulation, we use 30 minutes for C_{gossip} (simulation time units), 10 for L_{gossip}, and 50 for *view-size*, as this setting provides moderate average number of friends with acceptable network traffic.

4.6.3 Friendship Establishment

In this experiment, we investigate the effect of friend establishment on the performance of F2Frec over the respective metrics. We use 1 for the TTL of the query to measure the quality and effectiveness of friendship establishment. Also, query is forwarded to each friend *v* that is useful to query, in order to measure the quality and effectiveness of 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 4.5. In addition, we investigate the effect of friendship establishment on the performance of F2Frec over the respective metrics.

Table 4.3 shows the maximum results obtained after 24 hours of running the F2Frec system. 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.

Table 4.3. Results obtained by F2Frec over the respective metrics

α	Max. recall	Max. precision	Max. Com. cost	Max. Hit-ratio	Max. Avg. background traffic (bps)	Max. Avg. Friend
0	0.31	0.41	20	0.61	12.4	49.7
0.3	0.58	0.43	38.3	0.94	17.4	141.1
0.5	0.67	0.44	46	0.977	19	177.6
0.7	0.67	0.45	47	0.98	18.7	177.6
1	0.73	0.47	46.5	0.98	18.5	174.6

In Figure 4.5, we show the variation of average number of friends, recall, precision and hit-ratio 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 (Figure 4.5(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.

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 (Figure 4.5(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.

Figure 4.5(c) shows that precision starts at around 0.47 value and then decreases until stabilizing around 0.4 after 11 hours. Initially, a user *u* might have a few useful friends in its FOAF file. As gossip is used, and friendship establishment is performed, more useful friends are added at *u*'s FOAF file. Accordingly, more useful friends might be visited, and more documents might be recommended, and thus more irrelevant documents might be returned.

In Figure 4.5(d), we observe that the hit-ratio starts at low value and keeps on growing as the network size grows and becomes stable after 17 hours. Note that, at the beginning, a user may not have live (online) friends in its FOAF satisfying its query, and thus is not able to forward the query.

To conclude, we observe that combining users' usefulness with friend networks increases the likeness between users. Thus more new friends are added to users' FOAF files, which increase the recall, communication cost, hit-ratio, and background traffic. Also, we observe that the recall and precision are inversely related, and they depend on the length of the returned recommendation documents. If more documents are recommended, then more relevant documents might be recommended, which increases the recall value. On the other hand, more irrelevant documents might be included in the returned documents, which decreases the precision value.

In practice, if recall is preferred, importance is given to retrieve all or most of the relevant documents. In the other hand, if precision is preferred, importance is given to retrieve some of the relevant documents without going through a lot of irrelevant documents. In that case, it is important to rank the recommendation documents, return the top-k recommendation documents, and ensure as possible that the top-k returned documents are relevant (see Section 4.6.5).

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.

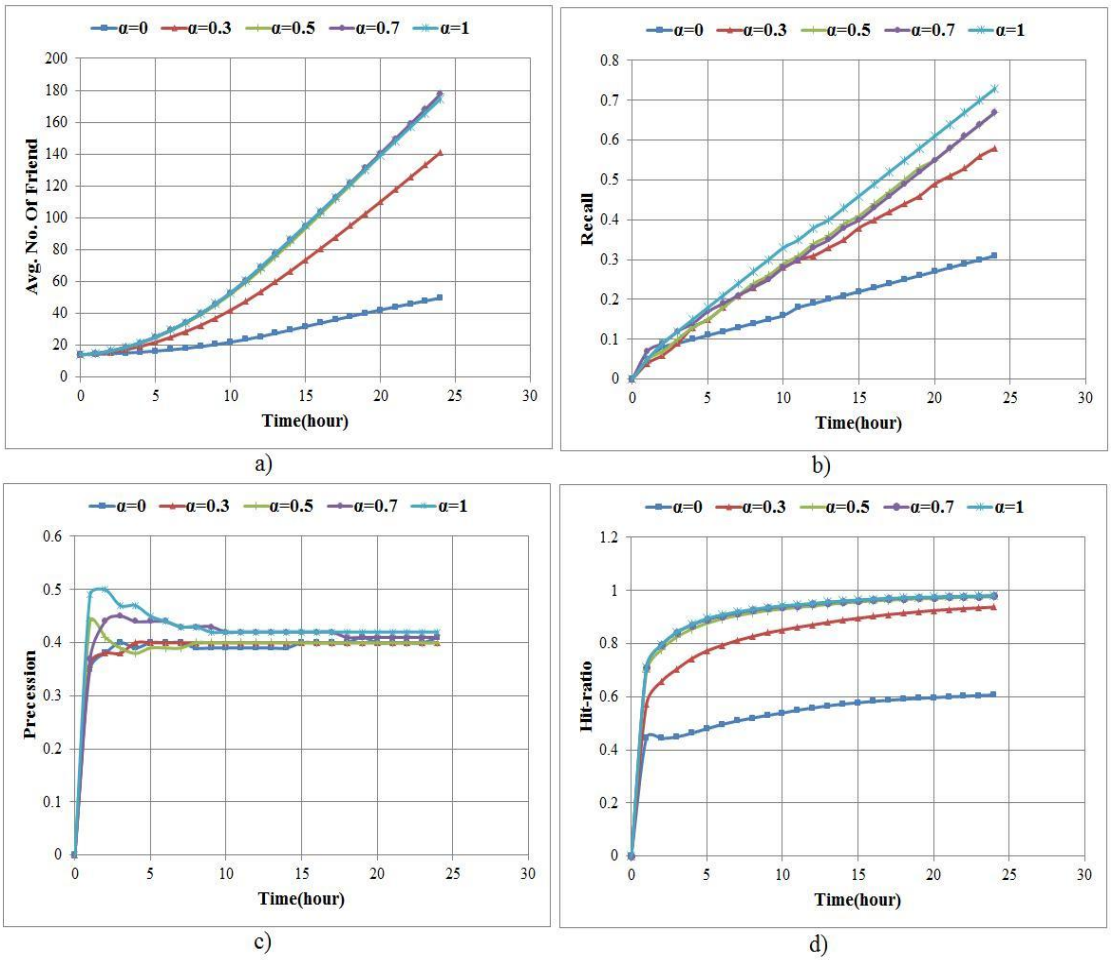


Figure 4.5. F2Frec performance over respective metrics

4.6.4 Impact of the Top-k Query Routing Algorithm

In this experiment, we investigate the effect of the top-k algorithm used to route a query over the recall metric. We use 1 for TTL and forward query to the top-10 adequate friends. In this experiment, we consider that each query $q \in Q$ has at most 5 topics i.e., $|T_q|$=5. For each query q, we rank the component of its V_q, and set the top-n components as the topics T_q of the query. Notice that the number n is the size of a query q*'s* topics i.e., $n = |T_q|$, and is selected randomly. We use more than one topic for each query q, to measure the effectiveness of the usefulness metric.

When a user u submits a query q, the top-k adequate friends are chosen by letting u's selects all friends that are useful for q based on the Equation 4.7, and adds them into a temporary list, denoted by *nominatList*. After that, u varies the way used to rank the adequate friends, and selects the top-10 to serve its query q as follows. First, u ranks each friend $v \in$ *nominatedList* based on the trust between u and v, and then u forwards q to the top-10 trustful friends. Second, u ranks each friend $v \in$ *nominatList* based on the usefulness of v for q, and then u forwards q to the top-10 useful friends for q. Finally, u ranks each friend $v \in$ *nominatList* based on the usefulness of v for q, and the trust between u and v, then u forwards q to the top-10 useful friend for q and trustful.

Table 4.4 shows the maximum results obtained from the experiments after 24 simulation hours. We observe that forwarding query q to the top-k useful friends for q, produces higher recall. The friends that are most useful for q, they have more relevant topics similar to q's topics. Thus, they have more documents related to q. Accordingly, many relevant documents are returned, which increases the recall value. Ranking friends based on the trust, produces a lower recall. There is no guaranty that the highest trustful friends have many relevant topics similar to q's topics. Accordingly, a few relevant documents are returned, which reduces recall.

When we rank friends based on trust and usefulness, the recall is moderate between both usefulness and trust, the first may return a high number of relevant documents while the other may return a few number of relevant documents. Therefore, the recall is higher than in trust but lower than in usefulness.

Table 4.4. Impact of the top-k routing algorithm

Top-10	Max. recall
trustful	0.163
useful	0.203
trustful + useful	0.187

To conclude, we observe that forwarding query to the top-k useful friends, increases the chance of returning more relevant documents. But forwarding query to

top-k trust friends increases the chance of returning more trusted and confidence recommendations.

4.6.5 Trade-off of Ranking Recommendations

In this experiment, we investigate the effect of ranking recommendations over the quality of recommendations. We vary the value of parameters a, b and c between 0 and 1, in order to investigate the trade-off of semantic similarity, popularity and the distance an trust based on the Equation 4.11. In each experiment we set the value of one parameter (a, b, or c) to 1 and the other two values to 0, except the last experiment, we set all parameter values to 0.33 i.e., a=b=c=0.33. In addition, we use 1 for the TTL of the query, and query is forwarded to each friend *v* that is useful to query.

In each experiment, when a user *u* initiates a query *q*. First, we generate recommendations for *q*. Then, we rank the recommendations based on: 1) semantic similarity, 2) popularity, 3) distance and trust between query's initiator and responders, 4) combinations of all. Next, we compute the recall and precision of the *q* when top-5, top-10 and top-20 recommendation documents are selected. Finally, we average the recall and precision values for all evaluated queries.

In this experiment, in addition to the classical *precision* that we have discussed in Section 4.6.1, we use a new metric, denoted by *RankPower,* to reflect the count (number) and rank (position) of the relevant documents recommended. RankPower has been first proposed by Meng et al. [103], but it has a lower bound of 0.5, and it is obtained when all the returned documents in the rank list are relevant i.e., the minimum value indicates the optimal performance. A revised definition of RankPower has been proposed by Tang et al. [149] and takes values between 0 and 1, where the value 0 indicates the worst performance and the value of 1 indicates the best performance. RankPower combines the count and the rank of the relevant documents appeared in the rank list, and is defined as follow.

$$RankPower = 100.\frac{n(n+1)}{\sum_{i=1}^{n} rank(i)} \quad (4.14)$$

Where *n* and *rank(i)* is the number of relevant documents and the rank of the *i*th relevant document appear in the top-k recommended documents, respectively.

Table 4.5 lists the maximum results obtained from the experiments after 24 simulation hours, in terms of precision and RankPower. We observe that ranking recommendation documents based on the semantic similarity produces the highest precision (0.78 when top-5 recommendation documents are selected) among all ranking methods proposed. This is due to the fact that, the documents that is most similar to a query *q*, most probably is relevant to *q*. Using distance and trust produces the lowest precision (0.45 when top-5 recommendation documents are selected).

In the other hand, using document popularity produces a precision higher than in distance and trust but lower than in semantic similarity (0.53 when top-5 recommendation documents are selected). While combining semantic similarity, with popularity and distance and trust produces a moderate precision (0.72 when top-5 recommendation documents are selected).

Similarly, the highest RankPower is obtained, when the recommendations are ranked based on the semantic similarity, because in that case, more relevant documents are appeared in the top-k recommendation documents. From the RankPower values, we observe that ranking recommendations methods put the relevant documents in the top positions of the rank list i.e., the relevant documents have higher ranks than the irrelevant documents.

RankPower gives us an idea of the positions of the relevant documents in the rank list, while precision gives us an idea of how many relevant documents are returned. For instance, the precision value of 0.78 obtained, when recommendations are ranked based on the semantic similarity, and the top-5 recommendation documents are selected, implies that almost 4 documents out of 5 are relevant. The RankPower value of 0.84 obtained from the same example, implies that the probability of finding the relevant documents in top positions of the rank list is 0.84. Thus, the user does not need to go through a lot of irrelevant documents to find the relevant documents in the rank list.

From Table 4.5, we observe that increasing the value of k decreases the precision and RankPower values in all ranking methods. This is due to the fact that, as more documents are recommended, more irrelevant documents might be returned.

To conclude, we observe that ranking recommendations based on the semantic similarity between documents and query increases the chance of including the relevant documents in the top-k selected documents. However, using the distance and trust, increases the confidence and trust of the recommendation documents, because these documents are recommended from trusted friends. Also, we observe that ranking recommendations methods give higher ranks to the most relevant documents than irrelevant documents, and put them in the top positions of the rank list.

Table 4.5. Varying the way used to rank recommendations

Parameters a, b, and c values	Top-5		Top-10		Top-20	
	Max. precision	Max. Rank-Power	Max. precision	Max. Rank-Power	Max. precision	Max. Rank-Power
a=1, b=c=0	0.78	0.84	0.75	0.81	0.72	0.80
b=1, a=c=0	0.53	0.76	0.51	0.75	0.50	0.71
c=1, a=b=0	0.45	0.72	0.43	0.71	0.42	0.68
a=b=c=0.33	0.72	0.78	0.69	0.74	0.66	0.72

4.6.6 Users' Relevant Topics of Interest Dynamism

As described in Section 4.5, users' entries are updated by *self-promotion* and *friend-promotion*. In this experiment we compare *friend-promotion* with *self-promotion* on the freshness of users' entries. In this experiment we measure the number of gossip cycles needed to refresh the users' *local-view*. The experiments are achieved as follows. 1) We run the F2Frec algorithms for a period of time until most of the users in the network become online. 2) We change the topics of interest for all users in the network. 3) We collect the number of gossip cycles needed to update all users' *local-views* i.e., all users' *local-views* become fresh and up-to-date.

From Figure 4.6 we observe that *friend-promotion* speeds up the freshness of users' *local-views*. In *friend-promotion* more than 50% of users refresh their *local-view* in less than 23 gossip cycles. *Friend-promotion* needs 50 cycles to fresh all users' *local-views*. *Self-promotion* needs 40 cycles to refresh 50% of users' *local-views*, and 70 cycles to updates all users *local-views*. However, *friend-promotion* increases the back ground traffic by a factor of 4.6, compared to *self-promotion*.

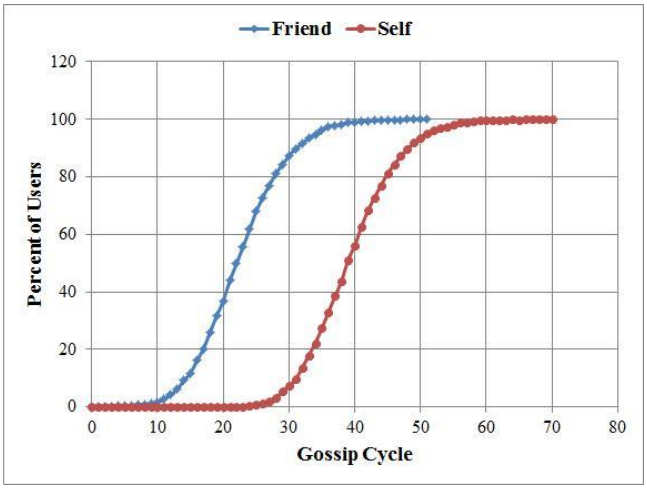


Figure 4.6. Fresh users' *local-views* vs. gossip cycles

4.7 Conclusion

In this chapter, 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.

We did an extensive experimental evaluation, using the Wiki vote social network and the TREC09 dataset. In this evaluation, we made the following observations. We showed that using small C_{gossip} and large L_{gossip} increases the number of friends added to users' FOAF files, because more new relevant users are explored and suggested for friendship. But, it increases the background traffic consumption. Combining users' usefulness with friend networks increases the likeness between users, and the possibility of establishing new friendships. This increases the recall, trust and confidence of the recommendation documents.

Forwarding query q to the top-k useful friends for q increases the recall of the recommendation documents. This is because, the top-k useful friends for q maintains a lot of documents related to q. Forwarding q to the top-k trustful friends increases the trust and confidence of the recommendation documents, because they are recommended from trusted friends.

Ranking recommendations puts the relevant documents in the top positions of the rank list. Ranking recommendations based on the semantic similarity between query q and documents produced the highest precision and RankPower. This is due to the fact that, the document that is most similar to q, most probably is relevant to q.

Chapter 5 P2P-RS Prototype

Abstract. In this chapter, we describe our prototype of P2P-RS, which we developed to validate our proposal, in particular, P2Prec and F2Frec. We developed our prototype as an application on top of the Shared-Data Overlay Network (SON), an open source development platform for P2P networks. We first describe the architecture of P2P-RS on top of SON. Then, we describe the demonstration of P2P-RS's main services (installing the P2P-RS peers, initializing peers, gossiping topics of interest among peers, keyword querying for contents) using our prototype.

5.1 Introduction

To further test and validate the feasibility of our proposal, we have implemented a prototype in Open Source Software of P2P-RS, based on P2Prec and F2Frec, described in the previous chapters. The prototype is described at our web site http://www-sop.inria.fr/teams/zenith/p2prec/ and the prototype code can be found at http://gforge.inria.fr/projects/p2prec/. The content of this chapter is partly based on our publications in [39][40].

P2P-RS is implemented in Java as an application on top of SON (http://wwww-sop.inria.fr/teams/zenith/SON), an open source development platform for P2P network applications developed in the Zenith team. The main objective of SON is to offer an open source development platform for P2P applications, by hiding the complex aspects of asynchronous messages between peers, thus helping developers building P2P applications rapidly. The developers do not need to deal with the complex distributed programming aspects; they only write the code logic for the behaviors of the peer (as in a simulator). In contrast, using other P2P development platforms such as JXTA [72], Jtella [71] and Groove [57] much effort and time are required to successfully build P2P applications.

Figure 5.1 shows the layered architecture of P2P-RS over SON. The top layer is the application layer that provides the essential interfaces for users. This layer gives users the ability to interact with the system such as sending a keyword query, receiving a response for a query, reading, storing, rating a document, managing FOAF files and establishing new friendships. Under the application layer is the P2P-RS layer, which provides the functions, services and protocols that are necessary for generating and receiving recommendations. Under the P2P-RS layer is the SON layer, the P2P

overlay that is used to support P2P-RS. At the bottom is the Internet layer, which provides the physical connections between peers in the system.

The rest of this chapter is organized as follows. In Section 5.2, we briefly recall the main aspects of SON. Section 5.3 describes in more details the architecture of P2P-RS. In Section 5.4, we describe the demo of P2P-RS's main services (installing and initializing P2P-RS peers, gossiping topics of interest among peers, keyword querying for contents) using our prototype. Section 5.5 concludes.

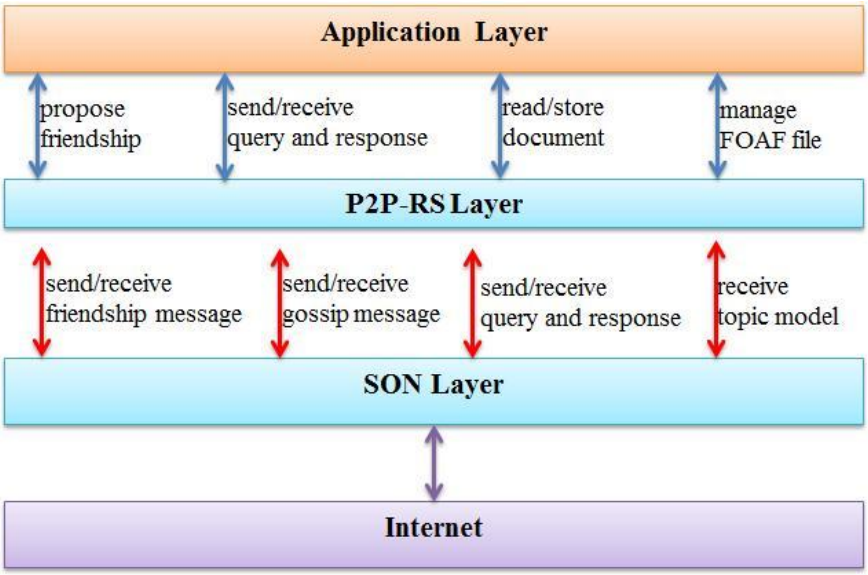


Figure 5.1. Layered architecture of P2P-RS

5.2 Shared-data Overlay Network (SON)

SON is an open source development platform for P2P networks using web services, JXTA and OSGi [112]. With SON, the development of a P2P application is done through the design and implementation of a set of components. A SON's component is a software package that encapsulates the code logic (typically generated by developers) that provides the functions and purposes of the component. It is accessed through interfaces that can be discovered at runtime. The interfaces comprise services provided by the component to other components in the system, as well as services required by the component to operate but implemented elsewhere. Given that these services are described through a machine-processable format, a component generator automatically generates the code of the services from their descriptions i.e., the developer does not deal with complex distributed programming aspects.

SON is implemented in Java on top of OSGi components that provide all basic services for the lifecycle of SON components, in particular, the deployment services. OSGi is a set of modules (called bundles) that provide services for Service Oriented Applications. Notice that several modules may provide the same service S, but at runtime, one module is chosen to provide the service S. The launching of a SON ap-

plication is defined through an OSGi configuration, which describes the application components.

The basic infrastructure of SON is composed of a Component Manager (CM), a Publishing and Discovery Component (PDC), and a Connection Component (CC) (see Figure 5.2). The PDC allows publishing or discovering components on different peers using a DHT. The CC provides connection between remote components on peers. The CM performs the creation of new component instances and the connections between them.

To establish a connection between two components A and B, the Component Manager uses the services description to associate the services provided by component A with the services required by component B, and conversely. After the connection process, the two components can communicate directly with each other, without going through the Component Manager.

The Component Manager delegates the management of lists of remote components to the Publishing and Discovery Component. In the current version, the OpenChord DHT implementation [111] is used for the Publishing and Discovery Component, although other implementations can be used. For this purpose, an interface has been defined with the usual methods (*put(key,value)* and *get(key)*) that can be expected from a DHT. At each creation of a component, the Component Manager publishes into this DHT the information for a remote component useful to connect to this component.

Notice that the DHT is used to publish and discover the components of SON, it does not have any coupling with the P2P overlays that are constructed for the applications build in top of SON. Moreover, we can use dedicated servers or DHT overlay from elsewhere to support the DHT overlay of SON.

The Connection Component is a component that handles the communication between remote components. It opens the TCP connection between peers. It is based on the concept of virtual pipes introduced by JXTA technology. This concept allows passing through a single TCP connection several logical communications (virtual pipes) between peers. Using this abstraction allows each component to open a virtual pipe to read messages sent to it.

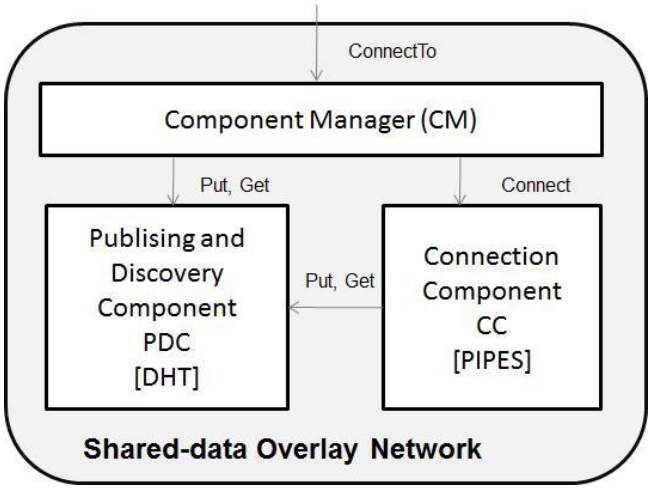


Figure 5.2. SON infrastructure

5.3 P2P-RS Implementation

We developed the P2P-RS as a SON application with two components: the *LDA component* for the document topics process and the *P2Prec component* for the recommendation process. The user interacts with those components with a graphical user interface.

In the following, we first present the P2P-RS's components and describe in details their implementation features. Then, we show the interaction between P2P-RS's components among different users in a network overlay.

5.3.1 P2P-RS Architecture

Figure 5.3 illustrates the P2P-RS implementation architecture. We used the Google Web Toolkit [56] to build the user interface. This toolkit allows defining a client/server application written completely in Java that runs in a web browser. It automatically compiles the Java client code into HTML and JavaScript, and easily permits to use Java libraries. Therefore, all graphical user interfaces are made of web pages and run in a classical browser.

Figure 5.3 shows that each component consists of a number of modules. For instance, the P2Prec component has four modules, *Data Management, Friend Establishment, Gossip Protocol* and *Query Processing*. Typically, each module is responsible for performing a set of activities. As an example, the gossip protocol module is responsible for providing the gossip behaviors of each user. Each component, module or user interface provides a set of interfaces that allow them to communicate between each other. These interfaces are either services or methods. We define two types of services, asynchronous and synchronous services, and one type of method, synchronous method.

- ***Asynchronous services***. These services are provided or required by P2P-RS's components. We refer to these services by the solid links that have a black arrow or red circle at their end. Hence, the component that is connected to the part ending with a circle provides the service, while the component that is connected to the part ending with an arrow requires the service. The components connected by a link that has two parts (black arrow and red circle) means that these components communicate locally. For instance, the P2Prec component provides the service *topicsOf(doc)* that it is required by the LDA component. The component connected to a link that has one part (black arrow or red circle) means that this service is provided to, or required by a remote instance of the same component. As an example, the P2Prec component provides/requires the *gossip(msg)* service to/from a remote P2Prec component.
- ***Synchronous services***. These services are used to pass messages between P2P-RS components and the graphical user interface. They are represented by the dash links that have two parts (green arrow and blue circle). The component connected to the blue circle provides the service to the user interface, while the component connected to the green arrow requires the service from the user interface. For ex-

ample, the service *propose(friend)* is required by the P2Prec component, while *submit(q)* is provided by the P2Prec component.

- ***Synchronous methods***. These methods are used to exchange information and data between a component's modules. They are represented by a link with an arrow at end. The direction of the link's row represents which module invokes the method. For instance, the method *get(user)* is invoked by the friend establishment module.

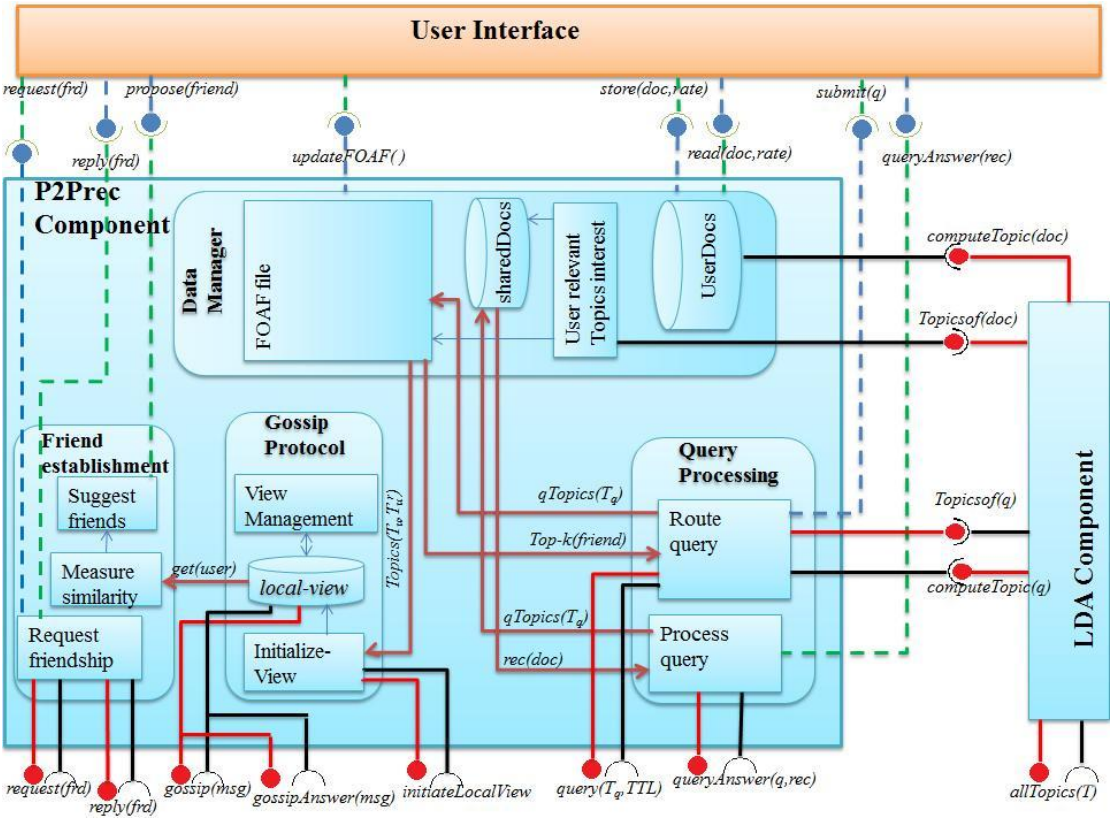


Figure 5.3. P2P-RS implementation architecture

5.3.2 P2P-RS's Components

We now present in more details P2P-RS's components including their modules, objectives, services, methods and functionalities.

5.3.2.1 P2Prec Component

The role of the P2Prec is computing, processing and generating recommendations. It has four modules:

1. **Data Manager**. This module manages user *u*'s documents and FOAF file. In addition, it is responsible for extracting *u*'s relevant topics of interest and maintaining the documents that are related to *u*'s relevant topics i.e., the documents that

the user *u* can provide recommendation for as described in Algorithm 3.1 (see Section 3.3.2). The data manager offers one asynchronous service:

- *topicsOf(doc)* returns the topic vector of a document (see Section 3.3.1). It is invoked by the LDA components, when LDA finishes computing the document topic vector.

The data manager offers two synchronous services:

- *updateFOAF(friend, topic, etc)* updates the FOAF file when provided with the fields to be added or deleted such as friends, topics of interest, etc. This service is invoked by the user interface.
- *store(doc,rate)* adds the document along with its rating value to user *u*'s documents, and it is invoked by user interface.

The data manager offers three synchronous methods:

- *topics(T_u,T_u^r, friends)* returns user *u*'s relevant (topics of interest) and friendship network. This method is invoked by gossip protocol in order to create *u*'s gossip entry.
- *top-K(friend)* returns the top-k useful and trust friends that can serve a query (see Section 4.4.1). This method is invoked by query processing when it routes a query.
- *rec(doc)* returns the documents of which similarity with a query exceeds a system-defined threshold (as described in Section 4.4.1). It is invoked by query processing in order to process a received query.

The data manager requires *read(doc, rate)* synchronous services to send a stored document to the user interface.

2. **Friend Establishment**. The role of this module consists in measuring the similarity between a user *u* and the users in its *local-view*, in order to suggest users to *u* for friendship establishing, and let *u* lunches a demand friendship with them, as depicted in Algorithm 4.1 (see Section 4.3.4). Friend establishment offers two asynchronous services:

   - *request(frd)* enables a user *u* to send a request message to a user *v* asking *v* for a friendship.
   - *reply(frd)* returns the reply message that has been sent by a remote user *v*. This service is invoked by the remote user *v*, when it is responding to a request friendship.

   Friendship establishment requires two synchronous services:

   - *propose(friend)* to send the suggested user to the user interface.
   - *reply(frd)* to send the response of a user *v* for friendship establishment to the user interface.

   Friendship establishment offers a *request(frd)* synchronous service that enables a user *u* to submit a request friendship to another user *v*. This service is invoked by the user interface.

3. **Gossip Protocol**. This protocol provides the gossip behavior of a user *u*, manages updates and initializes the *local-view* of the user *u* (see Section 3.4.3). It offers three asynchronous services:

   - *gossip(msg)* enables user *u* to submit a gossip message to remote user *v* when *u* initiates a gossip exchange. This service is invoked by *u*, the user who initiates the gossip exchange.
   - *gossipAnswer(msg)* returns the gossip message that has been sent by a remote user *v*. This service is invoked by the remote user *v* when it is responding to a gossip exchange.
   - *initiateLocalView(user)* adds an initial contacts to user *u*'s *local-view*. To do so, *u* invokes this service from bootstrap server BS.

   The gossip protocol has a *get(user)* synchronous method that returns the users that are currently maintained at *u*'s *local-view*. This method is invoked for friend establishment when the user *u* would like to establish new friendship.

4. **Query Processing**. This module performs query processing at a user *u* as described in Algorithm 4.2 (see Section 4.4.1). It routes the query submitted by *u* or received from a remote user *v* or processes a query received from a remote user *v*, and provides *v* with the recommendations. In addition, it ranks the recommendations that are returned as a response for a query that has been submitted by *u*. Query processing comes with three asynchronous services:

   - *topicsOf(q)* asynchronous service: returns the query topics vector when user *u* submits a keyword query. This service is invoked by the local LDA components when LDA finishes computing the query topics vector.
   - *query(Tq, TTL)* asynchronous service: enables user *u* to submit a query *q* to a remote user *v*. Notice that the query *q* either has been initiated by *u* or received from another remote user *x*. This service is invoked and provided by both *u* and remote user *v*.
   - *queryAnswer(q,rec)* asynchronous service: enables user *u* to receive a response for a query (initiated by *u*) from a remote user *v*. Also, it enables *u* to submit a response for the query (received from a remote user *x*) to a remote user *v*. This service is invoked and provided by both local user *u* and remote user *v*.

   Query processing provides the *submit(q)* synchronous service and requires the *queryAnswer(rec)* synchronous service:

   - *submit(q)* enables user *u* to submit a keyword query. It is invoked by the search form that is implemented in the user interface.
   - *queryAnswer(rec)* enables the query processing module to provide a response for a query to the user interface.

   Query processing offers two synchronous methods:

   - *qTopics(Tq)*: passes a query *q*'s topics T_q to data manager, in order to select the useful and trust friends that can serve *q*.

- *qTopicsVector(Vq, Tq)*: passes a query *q*'s topics T_q and the topic vector of the *q* to the data manager, in order to measure the similarity between *q* and each document that is related to *q*'s topics.

5.3.2.2 LDA Component

The LDA component performs local inference with LDA as described in Section 3.3.1. It downloads the topic model *T* from the bootstrap server. To do so, *u* invokes the *connect( )* service that is provided by the bootstrap server, which then returns the topic model *T* to *u* by invoking the *allTopics(T)* service that is provided by the LDA component. Then, LDA locally starts using the topic model *T* to extract the topic vectors of *u*'s documents and queries. The LDA component offers three asynchronous services:

- *computeTopics(doc)* enables user *u* to computes the topics hidden in its documents. This service is invoked by data manager when it starts extracting *u*'s relevant (topics of interest).
- *computeTopics(q)* gives user *u* ability to extract the topics of each query *q* submitted by *u*. It is invoked by query processing when *u* initiates *q*.
- *allTopics(T)* returns the topic model *T* from bootstrap server. This service is invoked by the LDA component at each user, and provided by the LDA component of the bootstrap server.

5.3.3 P2P-RS Components at Work

Now that we have introduced the P2P-RS components individually, we discuss how they work together in a network. We focus on the work of the P2Prec and LDA components. Moreover, we explain and describe the services and functions that are necessary for generating and receiving recommendations.

The services of the P2Prec component are the services for passive and active propagation through gossip services (*gossip* and *gossipAnswer* services) and the query services (*query* and *queryAnswer* services). There are two OSGi configurations (see Figure 5.4): the bootstrap server configuration and the Client (the peer) configuration.

To run the P2P-RS application, the bootstrap server must be started on a given machine (with a given IP address). This IP address will be used as the entry point into the P2P-RS network for new peers. At start-up time, a new peer must first identify itself with the bootstrap server (*connect* service) and the bootstrap server returns the current set of all topics (*allTopics* service). Then within the local peer's LDA component and the current topics, the topics of each document are computed locally.

After these steps, the peer can start the recommendation steps and documents discovery without any connection with the bootstrap server. Indeed, the search of topics of a new document (*computeTopic(doc)* service) and the computing of topics of a query (*computeTopic(query)* service) can be made locally with the local peer's LDA component. Depending on the evolution of documents on the P2P-RS network, the

bootstrap server may update the set of topics of documents, and inform the peers by broadcasting this new topic set (using the *allTopics* service).

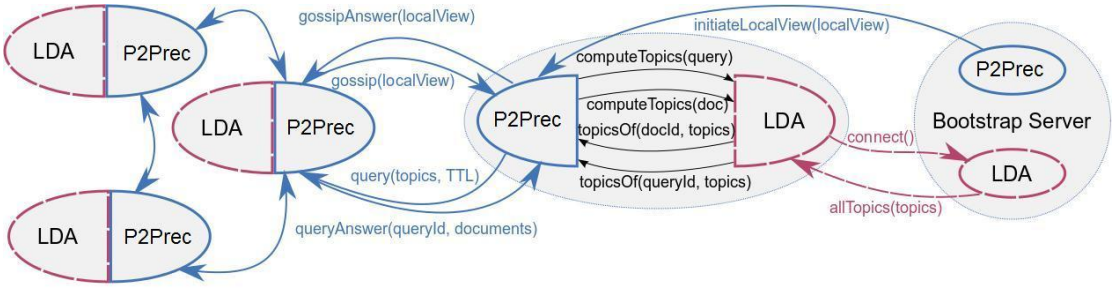


Figure 5.4. P2P-RS implementation at work

5.4 P2P-RS Demonstration

In this section, we describe how the P2P-RS's services cooperate using scenarios based on the Ohsumed documents corpus (MEDLINE) that has been used and discussed in Section 3.8.1.

We only implemented the interfaces that are necessary for demonstration to display the functions and services provided by the components of P2P-RS, which includes an interface that allows a user to submit a keyword query, and then displays the query response. In addition, we implemented an interface that shows the current friends of a user, and another interface that shows the entries of the current *local-view* of a user, and the user log gossip exchange. Furthermore, we have implemented the Rand gossip that is described in Section 4.3.2.

The demo with the MEDLINE information database (with a fixed data set) can be downloaded from the P2P-RS website (http://www-sop.inria.fr/teams/zenith/p2prec/index.php/Demo/) with the complete procedure for installing, deploying and running. In the following, we show how the application works, from the global installation to the utilization by an end-user.

5.4.1 Installation

In order to run a peer in the P2P-RS properly, any user (at a peer) has to connect first to the bootstrap server. Therefore we define a place the bootstrap server will run on. Every peer in the system will know its IP address. As the bootstrap server and any peer offer the same kind of services, we have defined two OSGi configurations for

running the P2P-RS's components: one as a bootstrap server, and one as a standard peer that will connect to the bootstrap server.

5.4.2 Initialization

Each peer consists of a LDA part coupled with a communication part (called P2Prec). As the demonstration starts, the bootstrap server is created, and so are several peers (100 of them). Each peer sends some of its documents, which are arbitrarily distributed among all peers, to the bootstrap server to perform LDA on a sample of all documents and to define the set of topics used in the network. Next, the bootstrap server informs all connected peers about the topics that are present in the network, and each peer indexes its own documents with the set of topics. Each peer is given an initial FOAF, which determines its friends in the network, and provides it information about them. It can now start gossiping with other peers, and the user belonging to the peer can send queries to discover documents. When connecting a new peer to the network, we show how it gets initial information in its FOAF file in two cases:

- It has already joined the network in the past (i.e. it knows other peers)
- It connects to the network for the first time.

5.4.3 Gossiping

The gossip service is at the heart of P2P-RS, and is transparent to the end-user. While peers exchange gossiping messages, the system recommends new friendships to users. For the sake of the demonstration, we developed an interface showing what is internally happening during gossiping (see Figure 5.5). The interface shows the current friends of the user, the gossiping messages sent and received by the peer, the gossip *local-view* that permits to find friends, etc. We show how the gossip mechanism notifies the user that other users share the same interests, and ask her to add them to her friend list.

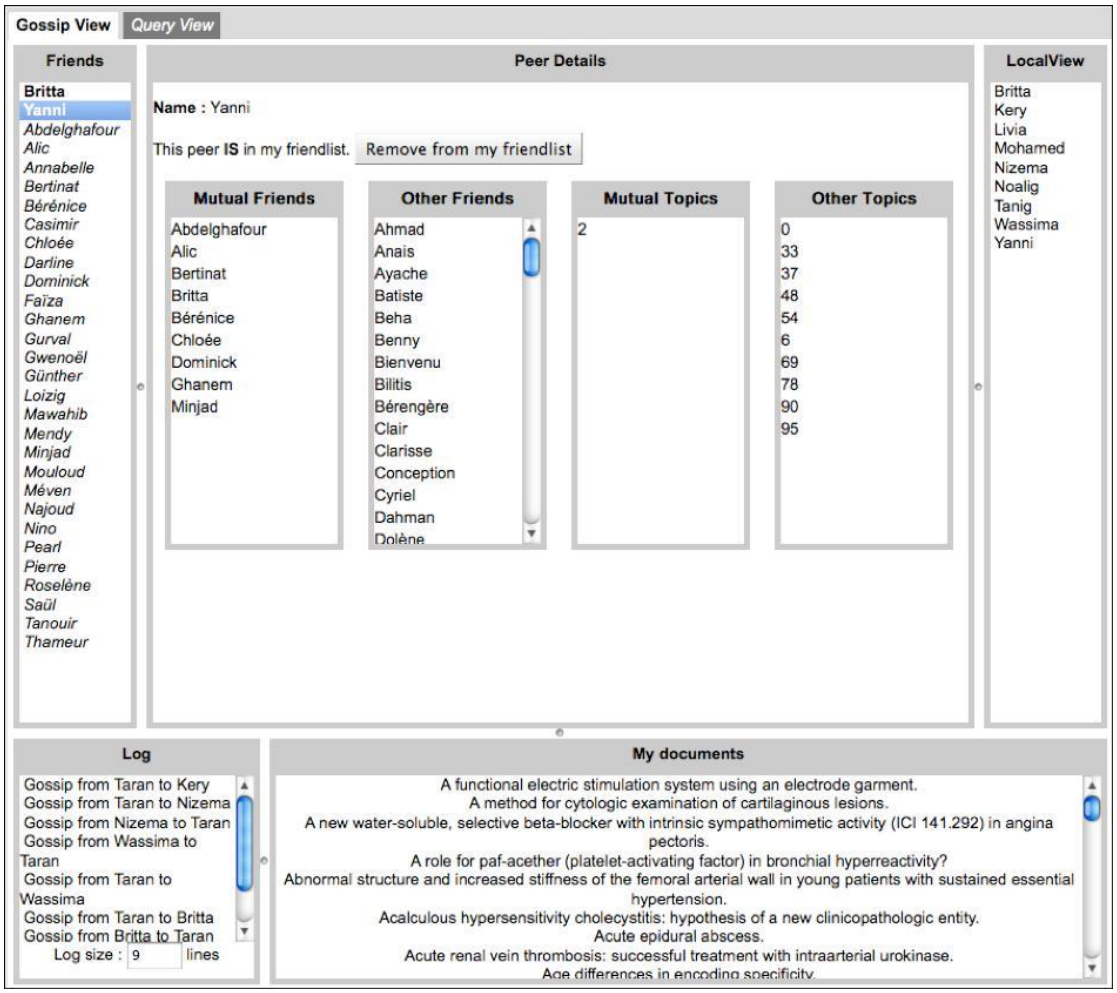


Figure 5.5. P2P-RS gossiping interface

5.4.4 Querying

Spreading information with gossip to make new friends has one aim: being able to answer queries accurately when a user searches for documents. This is where the query service is needed. The user is able to send a query for getting documents recommendations from her friends. The local LDA of the user translates the query into a set of relevant topics, and the peer sends them through the query service to the user's friends. Each friend may recommend documents depending on the similarity in terms of topics and the rate of the document. The query hops to friends of friends as many times as its TTL allows, the results being returned during the journey.

Figure 5.6 shows the result returned to the user after a query is sent. We show the results of the query for a user who has been in the network for a long time compared to a new user, and compare the accuracy and the number of answers she gets.

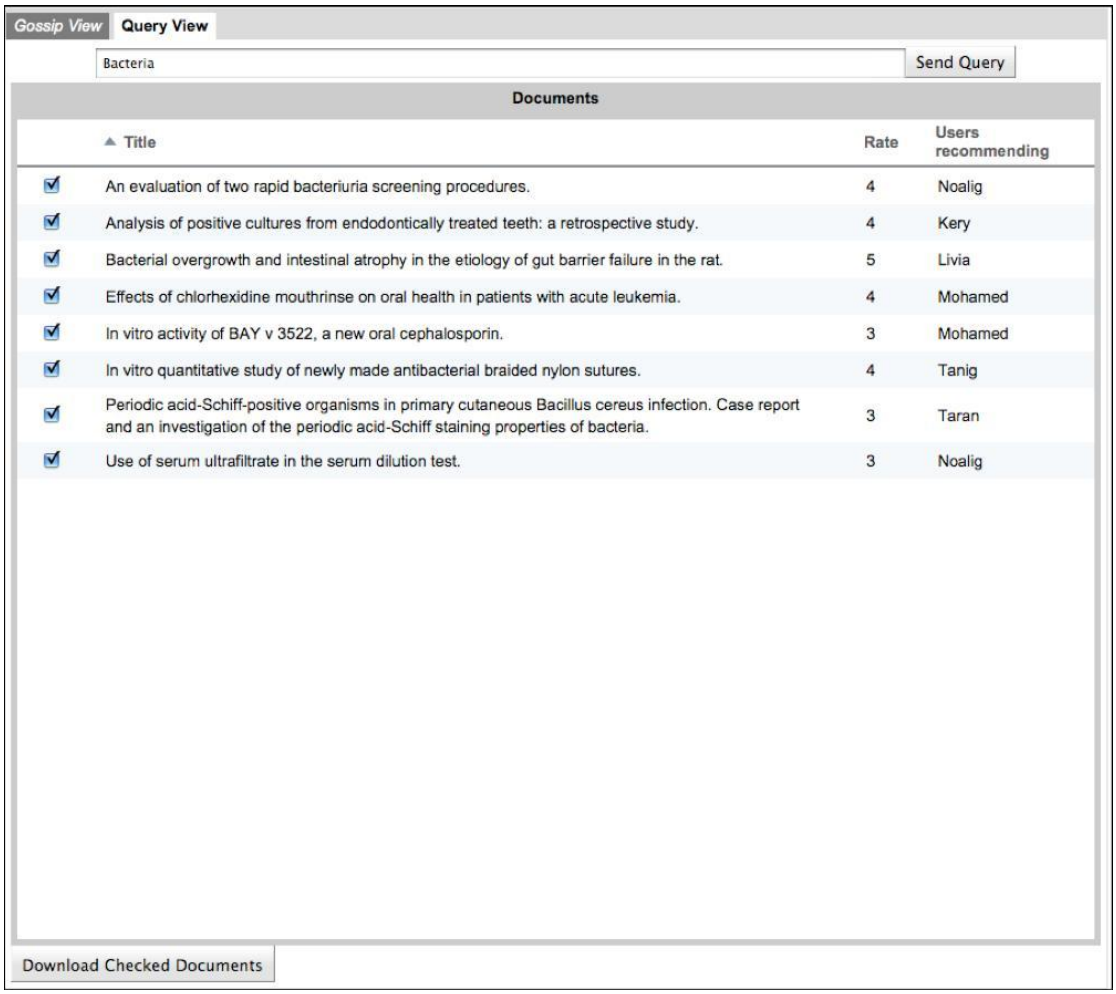


Figure 5.6. P2P-RS query interface

We can monitor the connections between the different peers of the network by using the SON's integrated graph view. Figure 5.7 shows an example of this view during the demo.

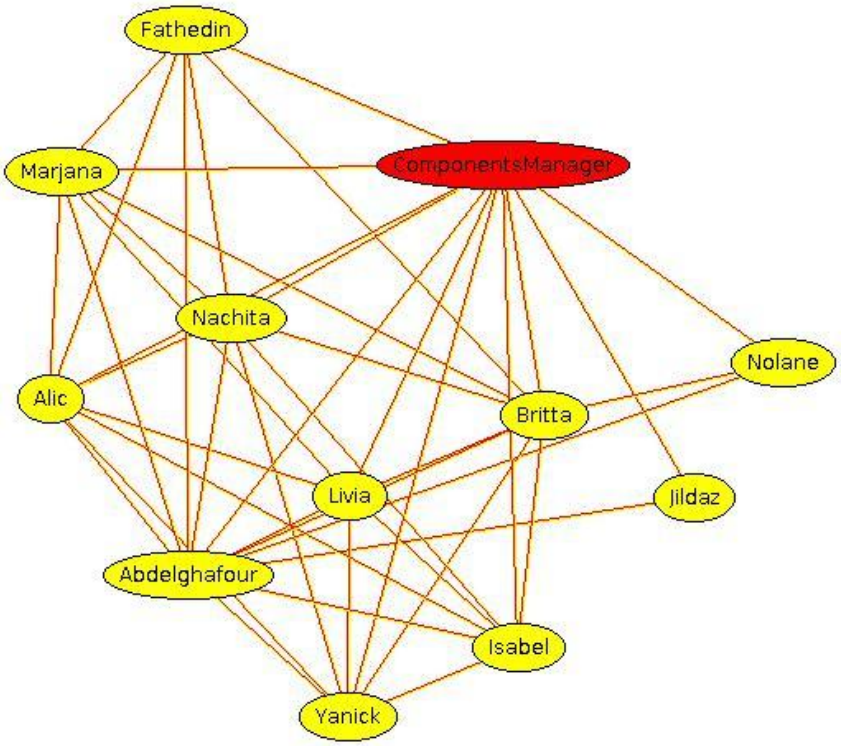


Figure 5.7. An example of the friendship graph of the P2P-RS demo

5.5 Conclusion

This chapter has described our prototype of P2P-RS, which we developed to validate our proposal, in particular, P2Prec and F2Frec. The P2P-RS prototype is implemented in Java on top of SON (http://wwww-sop.inria.fr/teams/zenith/SON), an open source development platform for P2P network applications developed in Zenith using web services, JXTA and OSGi. SON hides the complex aspects of asynchronous messages between peers, and thus makes it easier for developers to build P2P applications.

We showed the implementation architecture of P2P-RS using SON, with two components: the *LDA component* for processing document topics and the *P2Prec component* for the recommendation process. A full-fledge demonstration of this prototype has been built using the Ohsumed documents corpus, showing how friendship establishment, query processing, gossip protocol, etc. are involved.

This P2P-RS prototype implementation has been useful to validate our proposals, as well as to show its feasibility. Furthermore, using SON has been effective in making our development simpler and faster.

The P2P-RS web site is http://www-sop.inria.fr/teams/zenith/p2prec, the prototype code can be found at http://gforge.inria.fr/projects/p2prec, and the demo can be downloaded from http://www-sop.inria.fr/teams/zenith/p2prec/index.php/Demo

Chapter 6 Conclusion

Abstract. In this chapter, we summarize our main contributions and discuss future directions of research.

6.1 Summary of Contributions

This work has addressed the problem of decentralized recommendation in large-scale online communities. We proposed the design and implementation of a P2P recommendation system that exploits users' topics of interest and social data as parameters to construct and maintain a social P2P overlay, and generate recommendations. Our main contributions are as follows.

State-of-the-Art. We reviewed the techniques that have been proposed for building P2P recommendation systems. First, we introduced the main approaches of RSs namely *collaborative-filtering*, *content-based filtering* and *social-based filtering* along with their limitations. Then, we gave an overview of the main classes of P2P systems (*unstructured, structured* and *dynamic*), and highlighted the requirements that are needed to design P2P recommendation systems. Finally, we discussed the existing approaches for *P2P content management systems* (the systems that use information retrieval techniques to index and retrieve contents), and the approaches for *P2P prediction systems* (systems that are based on users' preferences).

P2Prec. We proposed the design of a scalable and reliable P2P recommendation system, called P2Prec. P2Prec leverages collaborative- and content-based filtering recommendation approaches. P2Prec uses relevant users to guide recommendations, given that relevant users are defined based on users' topics of interest. Users' topics of interest are automatically extracted from the contents the users hold using LDA. P2Prec uses the dynamic P2P overlay, where semantic-based gossip protocols are used to constructs and maintains the overlay, and to disseminate relevant users and topics of interests in the overlay. In addition, P2Prec uses an efficient query routing algorithm that uses users' gossip views to select the best relevant users that can serve query. Through an extensive performance experimentation (through simulation), we showed that using semantic gossiping increases recall and hit-ratio, because each user maintains more relevant users that can serve its queries in its gossip' view. Our results demonstrate that exploiting contents semantics and gossip protocols are perfectly adapted to the context.

The initial proposal of P2Prec was published in [36], and an improved proposal appeared in [38].

F2Frec. To increase the quality, confidence and trust of recommendations, we proposed the design of F2Frec, a P2P-RS that leverages content- and social-based filtering recommendation approaches by maintaining a P2P and friend-to-friend network. F2Frec uses users' relevant topics of interest and friendship network to suggest new friends that are useful to provide recommendations. Given that gossip protocol is used to disseminate users' relevant topics of interest, in order to find new interesting friends. F2Frec uses an efficient query routing algorithm that selects the top-k trust and useful friend to serve query. In addition, F2Frec ranks the recommendation documents based on the document popularity, the semantic similarity between the document and query, and the trust and topics of interest between the documents' responders and query's requestor. Simulation results showed that establishing new friends based on users' relevant topics of interest increases the recall, because more friends are added and thus more recommendations are returned. Establishing new friends based on trust increases the confidence of the result, as they returned from more trustable friends. Our extensive performance experimentation (through simulation) showed that ranking recommendations put the relevant documents in the top positions of the rank list. It also showed that ranking recommendations based on the semantic similarity between query *q* and documents produced the highest recall and precision. This is due to the fact that, the document that is most similar to *q*, most probably is relevant to *q*.

A partial version of this proposal appeared in [41].

Prototype. To fully validate our proposal, we implemented a prototype of P2P-RS for (http://www-sop.inria.fr/teams/zenith/p2prec/), which is based on P2Prec and F2Frec. P2P-RS is deployed as a standalone application on top of SON, an open source development platform for P2P networks developed in Zenith. SON hides the complex aspects of asynchronous messages between peers, and aids developers building P2P applications rapidly. We built a full-fledge demonstration of the P2P-RS prototype using the Ohsumed documents corpus, showing how friendship establishment, query processing, gossip protocol, etc. are involved.

The initial version of F2Frec is described in [39] and an improved version is described in [40].

6.2 Future Directions

In this section, we present a list of directions of research in P2P recommendation systems that we believe to be interesting to explore.

Document indexing and retrieval. In this work, documents and queries are represented by their topic vectors extracted from LDA. When a user *u* receives a query *q*, *u* measures the similarity between query *q* and each document *doc it* has locally, using the cosine similarity between V_{doc}=$[w_{doc}^{t_1}....w_{doc}^{t_d}]$ and $V_q = [w_q^{t_1}....w_q^{t_d}]$. Since *doc* and *q* are classified in the same topic, then most probably, they return a high similarity value. In that case, *doc* is included in the response of *q*, even though it is not rele-

vant to *q*. Typically, a topic includes a lot of documents, and most probably they are similar to *q* and will be included in *q*'s response. In fact, many of them are not relevant to *q*, even though they are similar to *q*, and this decreases the precision of the system. It would be interesting to use another indexing technique such Lucene indexing [10] to represent the documents and queries.

Experiments with different datasets. For this thesis, we have used the Ohsumed documents corpus, which consists of titles or abstracts from 270 medical journals over a five year period (1987-1991). In this dataset, documents (articles) are not associated with users. Moreover, there are no real ratings or feedbacks given by users of the documents in this dataset. Thus, in our experiments, we randomly distributed the documents over the users in the system, and generated a random rating between 0 and 5 for each document a user has. Accordingly, it is important to see if the behaviors of the system change when another dataset(s) are used. Also it is important to perform experiments with more realistic datasets. Moreover, it would be interesting to evaluate the performance of the system using human evaluations, which will give better insight of the performance of the system and users' satisfaction.

Ontology indexing. In our proposals, we use LDA to compute the topic model *T* in the system at a specific peer, e.g., the bootstrap server (see Section 3.3.1). The bootstrap server periodically aggregates a set of documents from the peers and estimates *T*. Each version of *T* is attached with a timestamp value. Each user *u* periodically contacts the bootstrap server and checks whether the timestamp of the topic model *T* it has is different from the one at the bootstrap sever. If that case, *u* downloads the new version of the topic model *T* from the bootstrap sever, and then computes its relevant topics of interest. Aggregating documents by the bootstrap sever, and continuously contacting the bootstrap server by users, increases the bandwidth consumed in the system, and introduces some type of centralization to the system. To handle this, users' topics of interest could be represented in terms of concepts (extracted from the documents they maintain) using a common defined ontology, thus providing more semantics that could be exploited for indexing contents. In that case, we would not need a bootstrap server to compute the topic model *T* in the system.

Recommendation in the Cloud. In cloud computing, services providers host a set of services at their infrastructures and deliver these services to customers over the internet. Then customers can use these services without installing and storing them in their personal computers. In the cloud, large amounts of data, content and knowledge are being spread over the service providers' infrastructures. Moreover, each provider has administrative control over the users' data in its infrastructure and there is neither communication nor interoperability between providers. To support data sharing among users that have different cloud providers, we could envision a decentralized recommendation system for multiple clouds.

Recommendation diversity. Diversity could be exploited as another parameter of recommendation. When a user submits a keyword query *q*, typically *q* includes a few numbers of keywords. Thus, only the documents that are most similar to *q*, in terms of *q*'s keywords are recommended. However, the user may also be interested in the documents that fit more general descriptions of *q*'s keywords. This can be done by extending *q*'s keywords (find the synonymy or the keywords that have some relations with *q*'s keywords) and including them in *q*'s body. Another way is to let the system

returns a few documents that are not similar to *q*, but have some relations (i.e., they have concepts related to *q*'s concepts, etc.) with *q*.

References

[1] Z. Abbassi, S. Amer-Yahia, L-V. Lakshmanan, S. Vassilvitskii, C. Yu. Getting recommender systems to think outside the box. In Proceedings of the 2009 ACM Conf. on Recommender Systems (RecSys), 285-288, 2009.

[2] G. Adomavicius, A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749, 2005.

[3] C. Aggarwal, J. Wolf, K. Wu, P. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In Proceedings of the 5th Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD), 201-212, 1999.

[4] R. Akbarinia, E. Pacitti, P. Valduriez. Reducing network traffic in unstructured p2p systems using top-k queries. Distributed and Parallel Databases, 19(2-3), 67-86, 2006.

[5] D. Alan, D-H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H-E. Sturgis, D-C. Swinehart, D-B. Terry. Epidemic algorithms for replicated database maintenance. In Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC), 1-12, 1987.

[6] Alexa-Top sites, 2012. http://www.alexa.com/topsites/

[7] S. Amer-Yahia, J. Huang, C. Yu. Jelly: A Language for Building Community-Centric Information Exploration Applications. In Proceedings of the 25th Int. Conf. on Data Engineering (ICDE), 1588-1594, 2009.

[8] S. Amer-Yahia, L. Lakshmanan, C. Yu. SocialScope: Enabling Information Discovery on Social Content Sites. In the 4th Biennial Conf. on Innovative Data Systems Research (CIDR), 2009.

[9] S. Amer-Yahia, C. Yu. Leveraging communities in social content sites. In Proceeding of 2009 EDBT/ICDT Workshops, 1-1, 2009.

[10] Apache Lucene Indexing web site. http://incubator.apache.org/lucene.net/

[11] C. Avery, R. Zeckhauser. Recommender systems for evaluating computer messages. Communications of the ACM, 40(3), 88-89, 1997.

[12] B. Baeza-Yates, B. Ribeiro-Neto. Modern information retrieval. Addison-Wesley Harlow, England, 1999.

[13] X. Bai, M. Bertier, R. Guerraoui, A-M. Kermarrec, L. Leroy. Gossiping personalized queries. In 13th Int. Conf. on Extending Database Technology (EDBT), 87-98, 2010.

[14] X. Bai, M. Bertier, R. Guerraoui, A-M. Kermarrec, L. Leroy. Personalized top-k processing: from centralized to decentralized systems. Ph.D. dissertation, 2010. http://tel.archives-ouvertes.fr/docs/00/54/56/42/PDF/Thesis_XiaoBai.pdf

[15] M. Balabanovic, Y. Shoham. Fab: Content-Based, Collaborative Recommendation. Communications of the ACM, 40(3), 66-72, 1997.

[16] M. Bawa, G-S. Manku, P. Raghavan. SETS: Search enhanced by topic segmentation. In Proceedings of 26th Int. Conf. in Information Retrieval (SIGIR), 306-313, 2003.

[17] D. Billsus, M-J. Pazzani. Learning collaborative information filters. In Proceedings of the 15th Int. Conf. on Machine Learning. Morgan Kaufmann, 46-54, 1998.

[18] BitTorrent P2P File Sharing. http://www.bittorrent.com/index.html/

[19] D-M. Blei, A-Y. Ng, M-I. Jordan. Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993-1022, 2003.

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

[21] A. Bonifati, G. Summa, E. Pacitti, F. Draidi. Semantic Query Reformulation in Social PDMS. CoRR abs/1111.6084, 2011. Submitted on 25 Nov 2011 to Data & Knowledge Engineering (DKE) Journal.

[22] N. Borch. Social Peer-to-Peer for social people. Proceedings of the Int. Conf. on Internet Technologies and Applications, 2005.

[23] J-S. Breese, D. Hecherman, C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the 14th Conf. on Uncertainty in Artificial Intelligence, 43-52, 1998.

[24] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In Proceedings of the 18th Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM), 126-134, 1999.

[25] R. Burke. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4), 331-370, 2002.

[26] J. Callan. *Distributed Information Retrieval*. In W. B. Croft, editor, Advances in Information Retrieval, Kluwer Academic Publishers, 127-150, 2000.

[27] M. Chevalier, C. Julien, C. Soulé-Dupuy. *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*. Information science reference, 2009.

[28] T. Chunqiang, Z. Xu, M. Mahalingam. psearch: information retrieval in structured overlays. Computer Communication Review, 33(1), 89-94, 2003.

[29] Citeulike web site. http://www.citeulike.org/

[30] A. Crespo, H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Stanford University, 2003. In the 3rd Int. Workshop, Agents and Peer-to-Peer Computing (AP2PC), LNCS, 3601, 1-13, 2005.

[31] M-F. Cuenca-Acuna, C. Peery, R-P. Martin, T-D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In Proceedings of the 12th IEEE Int. Symposium on High Performance Distributed Computing (HPDC), 236-246, 2003.

[32] S-C. Deerwester, S-T. Dumais, T-K. Landauer, G-W. Furnas, R-A. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science, 41(6), 391-407, 1990.

[33] Delicious web site. http://delicious.com/

[34] G. Derek, O-M. Donal. Instant Messaging & Presence management in Mobile Ad-Hoc Networks. In Proceedings of 2nd IEEE Conf. on Pervasive Computing and Communications Workshops, 55-59, 2004.

[35] R-L. Dice. Measures of the Amount of Ecologic Association between Species. Ecology 26(3), 297-302, 1945.

[36] F. Draidi, E. Pacitti, P. Valduriez, B. Kemme. P2Prec: a Recommendation Service for P2P Content Sharing Systems. Bases de Donnees Avancees (BDA), 26, 21-40, 2010.

[37] F. Draidi, E. Pacitti, P. Valduriez. Deliverable D5.2: demo of replication, caching and indexing services. DataRing Project, Dec. 2010.

[38] F. Draidi, E. Pacitti, B. Kemme. P2Prec: a P2P Recommendation System for Large-scale Data Sharing. Transaction on Large-Scale Data- and Knowledge- Centered Systems, LNCS, 6790(3), 87-116, 2011.

[39] F. Draidi, E. Pacitti, D. Parigot, G. Verger. Demo of P2Prec: a Social-based P2P Recommendation System. Journées Bases de Donnees Avancées (BDA), 27, 5-8, 2011.

[40] F. Draidi, E. Pacitti, D. Parigot, G. Verger. P2Prec: a Social-based P2P Recommendation System. Proceedings of the 20th ACM Conf. on Information and Knowledge Management (CIKM), 2593-2596, 2011.

[41] F. Draidi, E. Pacitti, M. Cart, H-L. Bouziane. Leveraging Social and Content-based Recommendation in P2P Systems. The 3rd Int. Conf. on Advances in P2P Systems (AP2PS), 13-18, 2011.

[42] F. Draidi, E. Pacitti, P. Valduriez. Deliverable D5.3: replication, caching and indexing services - experiments report. DataRing Project, Dec. 2011.

[43] M. El-Dick, E. Pacitti, R. Akbarinia, B. Kemme. Building a peer-to-peer content distribution network with high performance, scalability and robustness. Information Systems, 36(2), 222-247, 2011.

[44] eMule project. http://www.emule project.net.

[45] P-T. Eugster, R. Guerraoui, A-M. Kermarrec, L. Massoulieacute. Epidemic information dissemination in distributed systems. IEEE Computer, 37, 60-67, 2004.

[46] FaceBook web site. http://www.facebook.com/

[47] A. Fast, D. Jensen, B-N. Levine. Creating social networks to improve peer to peer networking. In Proceeding of the 11th ACM Int. Conf. on Knowledge Discovery in Data Mining (SIGKDD), 568-573, 2005.

[48] FastTrack web site. http://www.fasttrack.nu/

[49] W. Gaul, T. Schmidt-Thieme. Recommender systems based on user navigational behavior in the internet. Behaviormetrika, 29(1), 1-22, 2002.

[50] D. Gavidia, S. Voulgaris, M. Steen. Cyclon: Inexpensive Membership Management for Unstructured P2P Overlays. Journal of Network and System Management, 13(2), 197-217, 2005.

[51] Gnutella project. http://www.gnutelliums.com/

[52] J. Golbeck. Computing and Applying Trust in Web-Based Social Networks. Ph.D. Dissertation, University of Maryland, College Park, 2005.

[53] J. Golbeck, C-N. Ziegler. Generating Predictive Movie Recommendations from Trust in Social Networks. In Proceedings of the 4th Int. Conf. on Trust Management, iTrust, 93-104, 2006.

[54] D. Goldberg, D. Nichols, B. Oki, D. Terry. Using Collaborative Filtering to Weave an Information Tapestry. Communication of the ACM, 35(12), 61-70, 1992.

[55] K. Goldberg, T. Roeder, D. Gupta, C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. Information .Retrieval, 4(2), 133-151, 2001.

[56] Google Web Toolkit GWT. http://code.google.com/webtoolkit/

[57] Groove Peer Computing Platform, Groove Networks Inc. http://www.groove.net.

[58] GroupLens Research, MovieLens Data Sets. http://grouplens.org/node/12#attachments/

[59] J-L. Herlocker, J-A. Konstan, A. Borchers, J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In Proceeding of the 22nd Int. Conf. on Information Retrieval (SIGIR), 230-237, 1999.

[60] W-R. Hersh, C. Buckley, T. Leone, D-H. Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In Proceedings of 18th Int. Conf. in Information Retrieval (SIGIR), 192-201, 1994.

[61] W. Hill, L. Stead, M. Rosenstein, G. Furnas. Recommending and evaluating choices in a virtual community of use. Conf. Proceedings on Human Factors in Computing Systems, 194-201, 1995.

[62] HitWise Press Release, July, 11, 2006. http://www.hitwise.com/press-center/hitwise HS2004/social-networking-june-2006.php/

[63] J. Holliday, R. Steinke, D. Agrawal, A-E. Abbadi. Epidemic algorithms in replicated databases. IEEE Transactions on Knowledge and Data Engineering, 15(5), 1218-1238, 2003.

[64] A. Iamnitchi, M. Ripeanu, I. Foster. Locating data in (smallworld?) peer-to-peer scientific collaborations. In proceeding of the 1st Int. Workshop on Peer-to-Peer Systems (IPTPS), 232-241, 2002.

[65] A. Iamnitchi, I. Foster. Interest-Aware Information Dissemination in Small-World Communities. In Proceedings of High Performance Distributed Computing (HPDC), 167-175, 2005.

[66] M. Jelasity, A. Montresor. Epidemic-style Proactive Aggregation in Large Overlay Networks. In Proceedings of the 24th Int. Conf. on Distributed Computing Systems (ICDCS), 102-109, 2004.

[67] M. Jelasity, M. Alberto, B. Özalp. Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems (TOCS), 23(3), 219-252, 2005.

[68] M. Jelasity, B. Özalp. T-Man. Gossip-based overlay topology management. In Proceedings of the 3rd Int. Workshop on Engineering Self-Organizing Systems (ESOA), LNCS, 3910, 1-15, 2005.

[69] M. Jelasity, S. Voulgaris, R. Guerraoui, A-M. Kermarrec, M-V. Steen. Gossip-based peer sampling. ACM Transactions on Computer Systems (TOCS), 25(3), 8, 2007.

[70] H. Jin, N. Xiaomin, C. Hanhua. Efficient search for peer-to-peer information retrieval using semantic small world. In Proceedings of the 5th Int. Conf. on World Wide Web (WWW), 1003-1004, 2006.

[71] Jtella platform. http://polo.lancs.ac.uk/p2p/JTella/jtella.htm/

[72] JXTA project. http://jxta.kenai.com/

[73] G. Katherine, M. Hayden, R-V. Renesse, W. Vogels, K-P. Birman. GSGC: An efficient gossip-style garbage collection scheme for scalable reliable multicast. Technical report, Cornell University, Ithaca, NY, USA, 1997.

[74] D. Kempe, A. Dobra, J. Gehrke. Gossip-based computation of aggregate information. In Proceedings of the 4th IEEE Symposium on Foundations of Computer Science (FOCS), 482-491, 2003.

[75] J. Kendall, K. Kendall. Information delivery systems: an exploration of web pull and push technologies. Communications of the Association for Information Systems (AIS), 1(4), 1-43, 1999.

[76] P-B. Kenneth, H. Mark, O. Oznur, X. Zhen, B. Mihai, M. Yaron. Bimodal multicast. ACM Transactions on Computer Systems (TOCS), 17(2), 41-88, 1999.

[77] A-M. Kermarrec, M. Laurent, J-G. Ayalvadi. Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed systems, 14(3), 248-258, 2003.

[78] A-M. Kermarrec, M-V. Steen. Gossiping in distributed systems. Operating Systems Review, 41(5),2-7, 2007.

[79] A-M. Kermarrec, V. Leroy, A. Moin, C. Thraves. Application of Random Walks to Decentralized Recommender Systems. In proceeding of the 14th Int. Conf. in Principles of Distributed Systems (OPODIS), 48-63, 2010.

[80] H-J. Kim, J-J. Jung, G-S. Jo. Conceptual framework for recommendation system based on distributed user ratings. In Grid and Cooperative Computing, LNCS, 3032, 115-122, 2004.

[81] M. Kim, V. Raghavan. Adaptive Concept-based Retrieval Using a Neural Network. In Proceedings of Mathematical/Formal Methods in information retrieval Workshop at the 23th Int. Conf. in Information Retrieval (SIGIR), 33-40, 2000.

[82] R-A. King, A. Hameurlain, F. Morvan. Query Routing and Processing in Peer-To-Peer Data Sharing Systems. Int. Journal of Database Management Systems, Academy & Industry Research Collaboration, 2(2),116-139, 2010.

[83] I-A. Klampanos, J-M. Jose. Architecture for information retrieval over semi-collaborating peer-to-peer networks. In Proceedings of the ACM symposium on Applied computing, 1078-1083, 2004.

[84] J-A. Konstan, B-N. Miller, D. Maltz, J-L. Herlocker, L-R. Gordon, J. Riedl. GroupLens: Applying Collaborative Filtering to Usenet News", Communication of the ACM, 40(3), 77-87, 1997.

[85] S-R. Kruk, S. Decker, A. Gzella, S. Grzonkowski, B. McDaniel. Social semantic collaborative filtering for digital libraries. Journal of Digital Information, Special Issue on Personalization, 2006.

[86] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. Oceanstore: architecture for global-scale persistent storage. Proceedings of the 9th Int. conf. on Architectural support for programming languages and operating systems (ASPLOS), 190-201, 2000.

[87] L. Lacomme, Y. Demazeau, V. Camps. Personalization of a trust network. In Proceedings of the 1st Int. Conf. on Agents and Artificial Intelligence (ICAART), 408-415, 2009.

[88] N. Lathia, S. Hailes, L. Capra. Trust-based collaborative filtering. In Joint iTrust and PST Conf. on Privacy, Trust Management and Security (IFIPTM), 119-134, 2008.

[89] M. Laurent, E-L. Merrer, A-M. Kermarrec, A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In Proceedings of the 25th ACM Symposium on Principles of Distributed Computing (PODC), 123-132, 2006.

[90] L. Lee. Ranking Documents in Thesaurus-Based Boolean Retrieval Systems. Information Processing and Management: an Int. Journal, 30, 79-91, 1994.

[91] J. Li, B-T. Loo, J. Hellerstein, F. Kaashoek, D-R. Karger, R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In Proceedings of the 3rd Int. Workshop on P2P Systems (IPTPS), 207-215, 2003.

[92] J. Lianga, K. Rakesh, W-R. Keith. The FastTrack overlay: A measurement study. Computer Networks Journal, 50(6), 842-858, 2006.

[93] G. Linden, B. Smith, J. York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Internet Computing, 76–80, 2003.

[94] F. Liu, M. Li, L. Huang. Distributed information retrieval based on hierarchical semantic overlay network. In Proceedings of the 3rd Int. Conf. in Grid and Cooperative Computing (GCC), 657-664, 2004.

[95] LiveJournal social network. http://www.livejournal.com/

[96] Q. Lv, C. Pei, C. Edith, L. Kai, S. Scott. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th ACM Int. Conf. on Supercomputing (ICS), 84-95, 2002.

[97] J-C. Lv, X. Won-Goo. A pure peer-to-peer full text information retrieval system based on semantic overlay networks. In Proceedings of the 3rd IEEE Int. Symposium on Network Computing and Applications (NCA), 47-54, 2004.

[98] Market web site eBay http://www.eBay.com/

[99] S. Marti, Ganesan, Garcia-Molina. SPROUT: P2P routing with social networks. In Int. Workshop on Peer-to-Peer Computing and Data Bases, 425-435, 2004.

[100] P. Massa P. Avesani. Trust-aware collaborative filtering for recommender systems. In: Int. Conf. on cooperative information systems, LNCS, 3290, 492-508, 2004.

[101] B. Matthias, S. Michel, P. Triantafillou, G. Weikum, .C. Zimmer. Minerva: collaborative p2p search. In Proceedings of the 31th Int. Conf. on Very Large Data Bases (VLDB), 1263-1266, 2005.

[102] M-R. Mclaughlin, J-L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In Proceedings of the 27th Int. Con. on Information Retrieval (SIGIR), 329-336, 2004.

[103] X. Meng, Z. Chen. On User-oriented Measurements of Effectiveness of Web Information Retrieval Systems. Int. Conf. on Internet Computing, 527-533, 2004.

[104] S. Middleton, N. Shadbolt, D. Roure. Ontological user profiling in recommender systems. ACM Transactions on Information Systems, 22(1), 54-88, 2004

[105] S. Milgram. The small world problem. Psychology Today 1(1), 61-67, 1967.

[106] B-N. Miller, J-A. Konstan, J. Riedl. PocketLens, Toward a Personal Recommender System. ACM Transaction on Information Systems, 22(3), 437-476, 2004.

[107] MySpace web site. http://www.myspace.com/

[108] D. Nichols. Implicit rating and filtering. In Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering, 31-36, 1998.

[109] S. Ogasawara, C. Paulino, L. Gresta, P. Murta, C. Werner, M. Mattoso. Experiment Line: Software Reuse in Scientific Workflows. Scientific and Statistical Database Management (SSDBM), 264-272, 2009.

[110] Y. Ogawa, T. Morita, K. Kobayashi. A fuzzy document retrieval system using the keyword connection matrix and a learning method. Fuzzy Sets and Systems, 39, 163-179, 1991.

[111] OpenChord DHT project. http://open-chord.sourceforge.net/

[112] OSGi web site. http://www.osgi.org/

[113] T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. 3rd edition, Springer, 2011

[114] L. Page, S. Brin, R. Motwani, T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[115] A. Paul, P. Alex, R. Chirita, W. Nejdl, O. Scurtu. Knowing where to search: Personalized search strategies for peers in p2p networks. In Proceedings of the P2P Information Retrieval Workshop at the 27th Int. Conf. in Information Retrieval (SIGIR), 2004.

[116] M. Pazzani, D. Billsus. Learning and Revising User Profiles: The Identification of Interesting Web Sites. Machine Learning, 27(3), 313-331, 1997.

[117] Peersim p2p simulator. http://www.peersim.sourceforge.net/

[118] H. Peng, X. Bo, Y. Fan, S. Ruimin. A scalable p2p recommender system based on distributed collaborative filtering. Expert systems with applications, 27(2), 203-210, 2004.

[119] X-H. Phan. http://gibbslda.sourceforge.net/

[120] J. Pisson, T. Moors. Survey of research towards robust peer-to-peer networks: search methods. Computer Networks, 50(17), 3485-3521, 2006.

[121] J-A., Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D-H-J. Epema, M. Reinders, M-R. Van-Steen, H-J. Sips. TRIBLER: a social-based peer-to-peer system. Concurrency and Computation: Practice & Experience - Recent Advances in Peer-to-Peer Systems and Security, 20(2), 127-138, 2008.

[122] Y. Qiao, F-E. Bustamante. Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use. In Proceedings of the USENIX Technical Conf. (ATEC), 341-355, 2006.

[123] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A scalable content-addressable network. In Proceedings of the 2001 Conf. on Applications, technologies,

architectures, and protocols for computer communications (SIGCOMM), 161-172, 2001.

[124] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM Conf. on Computer supported cooperative work, 175-186, 1994.

[125] P. Resnick, H-R. Varian. Recommender systems. Communications of the ACM, 40(3), 56-58, 1997.

[126] S. Robertson, S. Jones. Relevance Weighting of Search Terms. Journal of the American Society for Information Science, 27 (3), 129-46, 1988.

[127] S. Robertson, D-A. Hull. The TREC-9 filtering track final report. The 9th Text REtrieval Conf. (TREC-9), 25-40, 2001.

[128] A. Rowstron, P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Middleware, 329-350, 2001.

[129] O-D. Sahin, F. Emek¸ci, D. Agrawal, A-F. Abbadi. Content-based similarity search over peer-to-peer systems. In Proceedings of Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P), 61-78, 2004.

[130] G. Salton. *Theory of Indexing*. Conf. Series in Applied Mathematics Society for Industry and Applied Mathematics, J. W. Arrowsmith Ltd., 1975.

[131] G. Salton, C. Buckley. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5), 513-523, 1988.

[132] G. Salton. *Automatic Text Processing*. Addison-Wesley, 1989.

[133] K. Sarda, P. Gupta, D. Mukherjee, S. Padhy, H. Saran. A distributed trust-based recommendation system on social networks. In 2nd IEEE workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2008.

[134] S. Saroiu, P-K. Gummadi, S. Gribble. Measurement study of peer-to-peer file sharing systems. In SPIE Multimedia Computing and Networking (MMCN), 2002.

[135] B-M. Sarwar, G. Karypis, J-A. Konstan, J-T. Riedl. Analysis of recommendation algorithms for e-commerce. In Proceedings of the 2nd ACM Conf. on Electronic commerce (EC),158-167, 2000.

[136] B-M. Sarwar, G. Karypis, J-A. Konstan, J-T. Riedl. Item based collaborative filtering recommendation algorithms. In Proceedings of the 10th Int. World Wide Web Conf. of ACM, 285–295, 2001.

[137] J-B. Schafer, J-A. Konstan, J-T. Riedl. E-commerce recommandation applications. Data Mining and Knowledge Discovery, 5(1-2), 115-153, 2001.

[138] A-I. Schein, A. Popescul, L-H. Ungar, D-M. Pennock. Methods and Metrics for Cold-Start Recommendations. In Proceeding of the 25th Int. Conf. in Information Retrieval (SIGIR), 253-260, 2002.

[139] U. Shardanand, P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In Proceeding of the ACM Conf. on Human Factors in Computing Systems, 210-217, 1995.

[140] A. Shepitsen, J. Gemmell, B. Mobasher, R. Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In Proceedings of the 2nd ACM Conf. on Recommender Systems, 259-266, 2008.

[141] S. Siersdorfer, S. Sizov. Social recommender systems for web 2.0 folksonomies. In Proceedings of the 20th ACM Conf. on Hypertext and hypermedia, 261-270, 2009.

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

[143] Slashdot web site. http://www.slashdot.org/

[144] I. Sola-Pool, M. Kochen, S. Milgram, T. Newcomb. *The Small World*. Ablex, Norwood, 1989.

[145] V. Spyros, M. Van-Steen. Epidemic-style management of semantic overlays for content-based searching. In Proceedings of the 11th Int. Euro-Par Conf. (EuroPar), 1143-1152, 2005.

[146] R. Sridharan, A-D. George, R-W. Todd, M-C. Chidester. Gossipstyle failure detection and distributed consensus for scalable heterogeneous clusters. Cluster Computing, 4(3),197-209, 2001.

[147] K. Sripanidkulchai, B-M. Maggs, H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In Proceedings of the 22nd Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM), 2003.

[148] I. Stoica, R. Morris, D. Karger, M-F. Kaashoek, B. Balakrishnan, A. Chord. A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 Conf. on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), 149-160, 2001.

[149] J. Tang, Z. Chen, A-W. Fu, D-W. Cheung. Capabilities of Outlier Detection Schemes in Large Databases: Framework and Methodologies. Knowledge and Information Systems, 11(1), 45-84, 2006.

[150] C. Tang, Z. Xu, S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In Proceedings of the 2003 Conf. on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), 175-186, 2003.

[151] Y. Teh, M. Jordan, M. Beal, D. Blei. Hierarchical Dirichlet processes. Journal of the American Statistical Association, 101(476),1566-1581, 2006.

[152] The friend of a friend (FOAF) project. http://www.foaf-project.org/

[153] The Grid'5000 web site. http://www.grid5000.fr/

[154] The PlanetLab web site. http://www.planet-lab.org/

[155] K-H-L. Tso-Sutter, L-B. Marinho, L. Schmidt-Thieme. Tag-aware Recommender Systems by Fusion of Collaborative Filtering Algorithms. In Proceeding of 23rd ACM Symposium on Applied Computing, 16-20, 2008.

[156] A. Tveit. Peer-to-Peer Based Recommendations for Mobile Commerce. Proceedings of the Int. Workshop on Mobile Commerce, 26-29, 2001.

[157] B. Upadhyaya, C. Eunmi. Social Overlay: P2P Infrastructure for Social Networks. In Proceedings of the Int. Conf. on Networked Computing and Advanced Information Management (NCM) 970-976, 2009.

[158] Y. Upadrashta, J. Vassileva, W. Grassmann. Social Networks in Peer-to-Peer Systems. Proceedings of the 38th Hawaii Int. Conf. on System Sciences (HICSS), 3-6, 2005.

[159] H. Uri, B. Shapira, P. Shoval. Information Filtering: Overview of Issues, Research and Systems. User Modeling and User-Adapted Interaction, 11, 203-259, 2001.

[160] J. Verhoeff, W. Goffman, J. Belzer. Inefficiency of the use of Boolean functions for information retrieval systems. Communications of the ACM, 4 (12), 557-558, 1961.

[161] C. Vicent, P. Felber, E. Biersack. Efficient search in unstructured peer-to-peer networks. In Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 271-272, 2004.

[162] J. Wang, J. Pouwelse, R-L. Lagendijk, M-J-T. Reinders. Distributive collaborative filtering for Peer-to-Peer file sharing systems. In Proceeding of ACM symposium on applied computing, 1026–1030, 2006.

[163] J. Wang, J-A. Pouwelse, J-D-V. Fokker, M-J-T. Reinders. Personalization of peer-to-peer television system. Multimedia Tool and Applications, 36(1-2), 89-113, 2008.

[164] Wikipedia the free encyclopedia. http://en.wikipedia.org/

[165] Wikipedia vote network. http://snap.stanford.edu/data/ wiki-Vote.html/

[166] R. Wilkinson, P. Hingston. Using the cosine measure in a neural network for document retrieval. In Proceedings of the 14th Int. Conf. in information retrieval (SIGIR), 202-210, 1991.

[167] B. Yann, A-M. Kermarrec. Proxsem: Interest-based proximity measure to improve search efficiency in p2p systems. In Proceedings of the 4th European Conf. on Universal Multiservice Networks (ECUMN), 62-74, 2007.

[168] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, H-P. Kriegel. Probabilistic memory-based collaborative filtering. IEEE Transactions on Knowledge and Data Engineering, 16(1), 56-69, 2004.

[169] B. Zhao, J. Kubiatowicz, A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report, U. C. Berkeley, 2001.

[170] C-N. Ziegler, S-M. Mcnee, J-A. Konstan, G. Lausen. Improving recommendation lists through topic diversification. In Proceedings of the 14th Int. Conf. on World Wide Web (WWW), 22-32, 2005.