Decentralizing news personalization systems

Antoine Boutet

ANNÉE 2013

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Antoine Boutet

préparée à l'unité de recherche INRIA
INRIA Rennes Bretagne Atlantique
Rennes 1

Decentralizing news personalization systems

Thèse soutenue à Rennes le 8 Mars 2013

devant le jury composé de :

Remerciements

Contents

CHAPTER 1

Introduction

The rapid evolution of the web has changed the way information is created, distributed, evaluated and consumed. The so-called Web 2.0 revolution has put users at the center of the web. They represent the greediest bandwidth consumers, the ultimate deciders of which applications are actually adopted and, maybe even more importantly, the most prolific content generators. Every single event, anywhere in the world, is likely to be instantaneously commented and debated on the Web by some community of users. The consequences are drastic. Even revolutions are now initiated on the Web, where new constitutions are being developed and debated on the fly. Not surprisingly, the temptation to get informed directly from the Web is bigger than ever and many of us succumb to it. Many spend a substantial amount of time browsing news on the Web but the quest is not always effective: the stream of available news is huge and keeps growing exponentially.

This thesis takes place in this context and tries to answer the following question: how can we filter out unwanted content while being able to discover new and relevant information? This question calls for recommendation systems for news personalization. While recommendation systems have been widely addressed in different directions since their introduction, the main challenge for news personalization is to deal with an increasing number of users and highly dynamic contents. Nowadays, two main techniques are widely used to provide personalization for users: (*i*) leveraging social networks and (*ii*) leveraging users and behavioral profiling.

Social networks such as Twitter and Facebook have become extremely popular and play a major role in the worldwide information dissemination [13, 15]. Originally created to keep track of friends, in most social networks, users declare explicit designated friends and share content and pictures with them. Furthermore, the majority of social platforms provide an interface to easily share content users have created or discovered in other websites or social applications with its network. Thus, it is possible for a user to recommend piece of news or information to its friends. In this model, users rely on a social network topology to receive personalized news and contents. However, relevant and important news often come from people outside one's explicit circle of friends and relying on explicitly declared friends strongly limits the content that can be received while potentially leading the spam. In addition, declaring someone as a friend one day does by no means qualify the notifications and news posted by that "friend" as highly relevant a few months later.

On the other hand, many modern websites or applications do not rely on explicit social networks, but collect and leverage information about users. The collected data can thus be exploited for different purposes. When users' actions and information come from participative online activities, this process is know as crowdsourcing. For instance, Digg lets users publish and express their opinions on content to identify the most popular ones. Such users information can be used to build users and behavioral profiling. For example, Facebook and Google leverage the buttons attached to websites offering an interface with their social networks to track users movements on the Web. These information allows them to increase the knowledge on users habits in order to increase the efficiency of search engines or the personalization system. This users information can also be exchanged through different companies to propose personalized advertisements [1]. The more a system is knowledgeable about the interests of its users, the more it is able to target relevant personalizations. However, this increase of monitoring scares users and may dissuades them from expressing opinions about their preferences, which, in turn, renders the entire point of a collaborative recommendation system useless.

While personalization greatly enhances the user experience, it raises a multitude of privacy challenges. For instance, personalized content in a social platform can reveal potentially embarrassing information directly to friends, family, and colleagues. Furthermore, users loose the control on their private data and how they are exchanged and exploited. Facebook, for example, commercially exploits user profiles information and have changed their terms of use since 2007 in order to mention that all content on their platform belong to them. More recently, end of 2012, Instagram has changed their terms of use to allow it to sell pictures from users without their agreement. Ownership and excessive commercial and unethical use of data raise more and more concerns about privacy from users.

Most recommendation systems are centralized. This architecture has the advantage to allow the provider to benefit from a global knowledge on the users behavior and the content. However, this model needs computationally intensive approaches. Only large companies can afford the cost of scaling with centralized architectures hosted by cloud solutions. Even when they do so, they confess their inherent difficulty in coping with dynamics [52] and they typically do off-line cluster-oriented filtering. In addition, a centralized architecture exhibits obvious vulnerabilities. Indeed, a system hosted by a set of servers can get attacked by some form of news bombing [] or meet infrastructure failures or scalability issues [14]. Furthermore, recent events such as in Tunisia, Egypt and Syrie have pointed out important limitations of centralized applications regarding Internet censorship. Censors are easily able to filter centralized social networks or video-sharing solutions in order to block political debate or to prevent the spread of some information.

On the other hand, peer-to-peer networks have emerged providing a highly-scalable network foundation upon which to build distributed applications. First peer-to-peer systems can provide a highly-scalable and a cheaper alternative than running a huge system on a cost and energy consuming datacenter. Effectively, the scalability model of datacenter usually comes from the number of resources a company has to dedicate to the system. Second and may be more importantly, peer-to-peer solutions might become an interesting alternative to Big brother kind of companies to provide a collective platform and privacy-aware information dissemination system. For all these reasons, we believe that it is of great interest to consider implementing a peer-to-peer news recommender.

Challenges

The main challenge of this thesis is to propose an effective distributed recommendation system for news personalization while respecting users privacy.

Contributions of this thesis

The motivation of this work is to explore the feasibility of a news personalization system which is user-centric and collaborative at its very architectural level. The goal is to design a solution where every node hosts and protects the interest profile of its associated user and dynamically collaborates with others to cluster interests and achieve an effective on-line personalized dissemination. In this context, P2P approaches are attractive because they naturally scale and circumvent a central entity that controls all user profiles and potentially exploiting them for commercial purposes. However, the absence of any global knowledge calls for CF schemes that can cope with partial and dynamic interest profiles. Designing an effective and privacy aware filtering scheme that is simple and lightweight enough to be easily deployable on personal machines with no help from any central server is challenging.

The first contribution of this thesis is a decentralized instant news recommender. For readability, in this first description, we do not address the privacy aspects but show that news personalization can be achieved by a decentralized system that could be effectively used for collaboratively filtering and disseminating instant news items in a fully personalized manner.

While decentralization removes the prying eyes of *Big-Brother* companies, privacy breaches can come from other users if the filtering schemes require personal tastes to be made public. The second contribution of this thesis is an obfuscation mechanism to preserve privacy while leveraging user profiles in distributed recommender systems.

A fully distributed news recommender systems has many benefits in term of failure resilience and scalability, and allows users to freely use it without any kind of constraints (*i.e.* advertisements). However, commercial websites and content editors have not yet found a business model adapted to this distributed architecture. The last contribution of this thesis explores a novel scheme leveraging the power of the distribution in a centralized architecture. This hybrid scheme democratize personalized systems by providing an online cost-effective scalable architecture for content providers at a minimal investment cost.

Contributions of this thesis are summarized in the Figure 1.1.

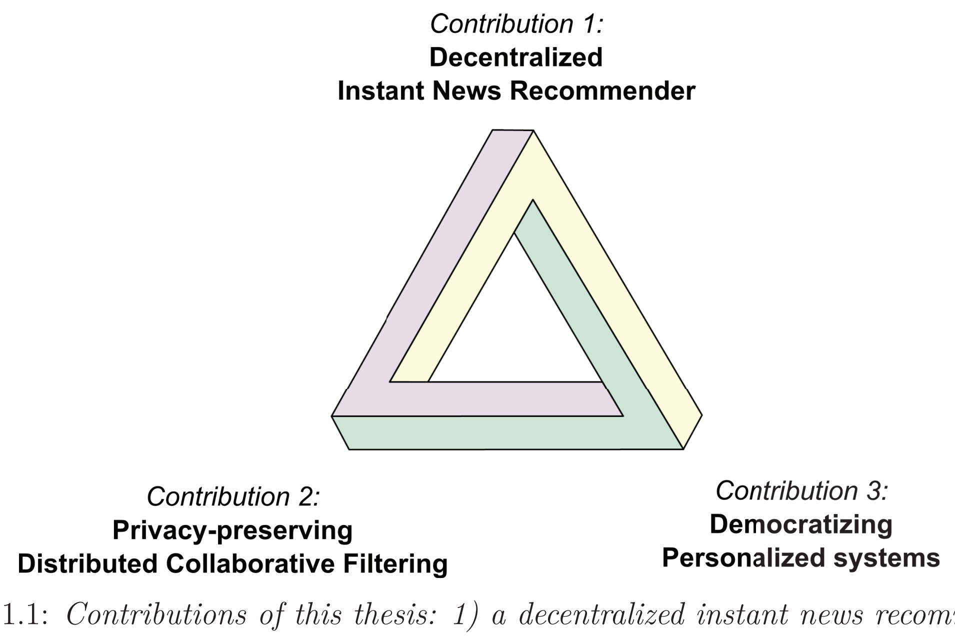


Figure 1.1: *Contributions of this thesis: 1) a decentralized instant news recommender, 2) mechanisms for privacy-preserving distributed collaborative filtering, and 3) a hybrid scheme for democratizing personalized systems.*

Other contributions

During my thesis, I have also conducted analysis on the users availability in online social networks and analysis and users classification in Twitter during elections. These contributions are not described in the document and are briefly described below.

The impact of users availability in OSNs [37]

Availability of users may have a big impact on online applications. Based on an availability trace collected from MySpace, we show that the online presence of users tends to be correlated to those of their friends. We then show that user availability plays an important role in some algorithms and focus on information spreading. Identifying central users *i.e.* those located in central positions in a network, is key to achieve a fast dissemination and the importance of users in a social graph precisely vary depending on their availability.

Characteristics of political parties and polarization of users in Twitter [38–40]

In modern politics, parties and individual candidates must have an online presence and usually have dedicated social media coordinators. In this context, we studied the usefulness of analysing Twitter messages to identify both the characteristics of political parties and the political leaning of users. As a case study, we collected the main stream of Twitter related to the 2010 UK General Election during the associated period. We examined the characteristics of the three main parties in the election and highlighted the main differences between parties. From these observations, we developed both an incremental and practical classification method which uses' tweet activity. The experimental results showed that the proposed classification methods achieved good classification accuracy and outperforms other classification methods that require expensive costs for tuning classifier parameters and/or knowledge about network topology.

Organization of the document

The rest of this document is organized as follows. Chapter 2 presents background and some relative works, and Chapters 3, 4, and 5 present our contributions 1,2 and 3, respectively. Finally, we draw some conclusions and set the perspectives of this thesis in Chapter 6.

Chapter 3: Decentralized Instant News Recommender

This contribution presents WhatsUp, a collaborative filtering system for disseminating news items in a large-scale dynamic setting with no central authority. WhatsUp constructs an implicit social network based on user profiles that express the opinions of users about the news items they receive (*like-dislike*). Users with similar tastes are clustered using a similarity metric reflecting long-standing and emerging (dis)interests. News items are disseminated through a novel heterogeneous gossip protocol that (*1*) biases the orientation of its targets towards those with similar interests, and (*2*) amplifies dissemination based on the level of interest in every news item. Evaluation shows that WhatsUp outperforms various alternatives in terms of accurate and complete delivery of relevant news items while preserving the fundamental advantages of standard gossip: namely, simplicity of deployment and robustness.

Chapter 4: Privacy-preserving Distributed collaborative filtering

This contribution presents a mechanism to preserve privacy while leveraging user profiles in distributed recommender systems typically in WhatsUp. Our approach relies on (*1*) an original obfuscation mechanism hiding the exact profiles of users without significantly decreasing their utility, as well as (*2*) a randomized dissemination algorithm ensuring privacy during the dissemination process. Our system is evaluated against an alternative providing differential privacy both during profile construction and dissemination. Results show that our solution preserves accuracy without the need for users to reveal their preferences. Our approach is also flexible and more robust to censorship.

Chapter 5: Democratizing personalized systems

Finally, we explore a novel scheme and presents *DeRec*, an online cost-effective scalable architecture for collaborative filtering personalization. In short, *DeRec* democratizes the recommendation process by enabling content-providers to offer personalized services to their users at a minimal investment cost. *DeRec* achieves this by combining the manageability of centralized solutions with the scalability of decentralization. *DeRec* relies on a hybrid architecture consisting of a lightweight back-end manager capable of offloading CPU-intensive recommendation tasks to front-end user browsers. The extensive evaluation of *DeRec* on reals workloads conveys its ability to drastically lower the operation costs of a recommendation system while preserving the quality of personalization compared to a classical approach.

CHAPTER 2

Background

We overview in this chapter the background and some related works connected to news personalization. We first address this problem through event notification systems, also called Publish/Subscribe systems. We focus on peer-to-peer architectures and more precisely on gossip-based solutions which provide a sound basis to achieve robustness and scalability. Second, we address personalization of news and recommendation systems. We introduce their goals and challenges, and we review some related works in term of algorithms and architectures.

2.1 Large scale event notification systems

Notifications play a prominent role in dissemination information today. Most of websites propose their content through RSS feeds, social networks provide API to easily get notification and publish messages, and recommendation systems send alerts to users when a new item matches their interests. Event notification usually relies on the so-called Publish-subscribe (pub/sub) communication paradigm [66]. In a classical pub/sub system, subscribers register their interest in an event, or classes of events, and are subsequently asynchronously notified of events generated by publishers. Many variants of this paradigm have been proposed, each variant being specifically adapted to some given application or network model [66].

Regarding the subscription and how subscribers' interest is expressed in relation to information, several schemes have been proposed with different degree of expressiveness. Pub/sub systems can be roughly classified into two categories, topic-based and content-based depending on the matching model. In a topic-based system, messages are published to "topics" and all subscribers to a topic will receive all messages published to this topic. In a content-based system, messages are only delivered to a subscriber if the content of those messages match constraints defined by the subscriber.

The traditional distributed architectures for pub/sub systems are based on a set of brokers and each client establishes affinity with a broker. Subscribers submit their subscriptions to their chosen broker and publishers send their events to their affiliated broker. In such systems, brokers need to advertise subscriptions to build routing state in the connected broker network in order to match the events against users subscriptions and to forward them to the users accordingly. Solutions such as Siena [47], Gryphon [133], Hermes [117] or Corona [123] and more recently Bluedove [110] are in this category.

Another type of architecture replaces the broker-based models with peer-to-peer overlays. Due to its inherent scalability, peer-to-peer systems are good candidates for event notification at large scale. In this section, we provide some background on peer-to-peer architecture and peer-to-peer pub/sub. Then, we focus on gossip-based approaches which provide good guaranty in term of robustness and scalability. Finally, we review some related works in this field.

2.1.1 Peer-to-Peer Pub/Sub

Peer-to-peer (P2P) is a distributed architecture that partitions tasks or workloads among peers. This differs from client/server architectures, in which some computers are dedicated to serving the others. Several P2P platforms such as Gnutella [3], Kazaa [7] or Skype [12] have demonstrated the scalability of the P2P paradigm.

This scalability stems from the capability of each peer to both benefits from the system as a client and contributes to the system as a server. As a result, the number of peers requesting the service increases accordingly to the number of peers which provide the service. They are self-organizing, coping naturally with nodes arrivals and departures (*i.e.* churn) and are resilient to failures. In such systems, peers are organized into a logical structure on top of the physical network, called an overlay network, using specific topologies such as rings, hypercubes, trees, random graphs, etc. These topologies define the neighbors for each node. No peer has a global knowledge of the system, based only on individual decisions, global properties on the system emerge. This is precisely what makes them both attractive and complex to maintain.

P2P networks are roughly classified as structured [125, 128, 151] or unstructured [87, 142]. Structured overlay networks require a joining node to follow a specific protocol that assigns it to a specific position within the overlay topology. Unstructured overlay networks allow nodes to occupy any position within the overlay network.

The basic idea behind P2P pub/sub is to organize nodes into an overlay network to efficiently disseminate events. Scribe [48] and Bayeux [152] are examples of pub/sub using a structured overlay built on top of Pastry [128] and Tapestry [151], respectively. In these approaches, a spanning tree is built for each topic, with a rendezvous node at the root which delivers the events to the nodes that join the tree. However, such systems force many nodes to relay events for which they have not subscribed as they happen to be on the path towards the rendezvous node.

In the following, we consider only gossip-based approaches which represent robust and scalable alternative.

2.1.2 Gossip-based approaches

A gossip protocol (or epidemic protocol) is a type of communication protocol inspired by the form of gossip seen in social networks or spreading of a disease in a population of individuals. Between the tree (efficient but fragile) and flooding (robust but inefficient) gossip protocols [57, 60] are well known to be simple, efficient, and robust in particular for event notification.

Gossip protocols

Gossip protocols involve periodic message exchanges between node pairs propagating local information. This allows users to refine and to maintain up to date their local version. The operations of a gossip protocol at a given cycle in a push/pull model are depicted in Figure 2.1. Each node runs two processes, an active thread which initiates communication (push) and receives peer state (pull), and a passive thread which mirrors this behavior. We use p to denote the peer who initiates the gossip (active thread) and p' to denote the peer who reacts to the gossip (passive thread). The fundamental operations driving a gossip protocol and influencing performance and reliability are (*1*) PeerSelection: to whom a peer gossips the message, (*2*) DataExchange: which information a node propagates in the network, and (*3*) DataProcessing: how a node merges the received information with its local version.

Properties of gossip protocols have been extensively studied [60], and have been used in many directions such as failure detection [124], aggregation [84], load balancing [85],

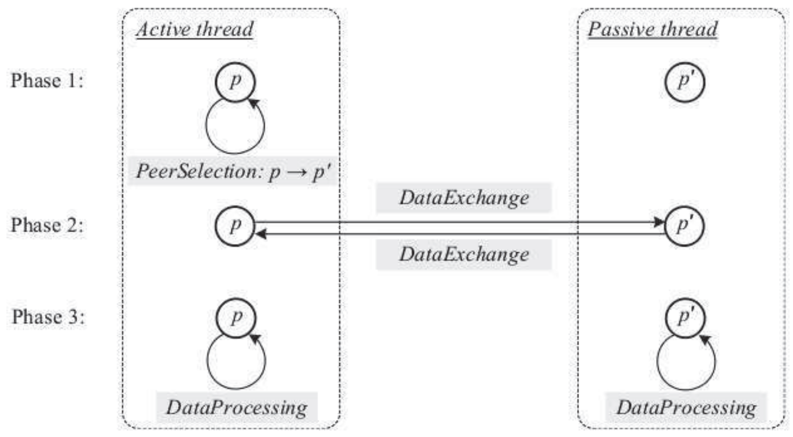


Figure 2.1: *A gossip operation between two peers [27].*

synchronization [111], streaming [69], or as in WhatsUp in topology management [87, 142, 143] and information dissemination [35, 60, 67, 68]. For this reason, we introduce their underlying concepts bellow.

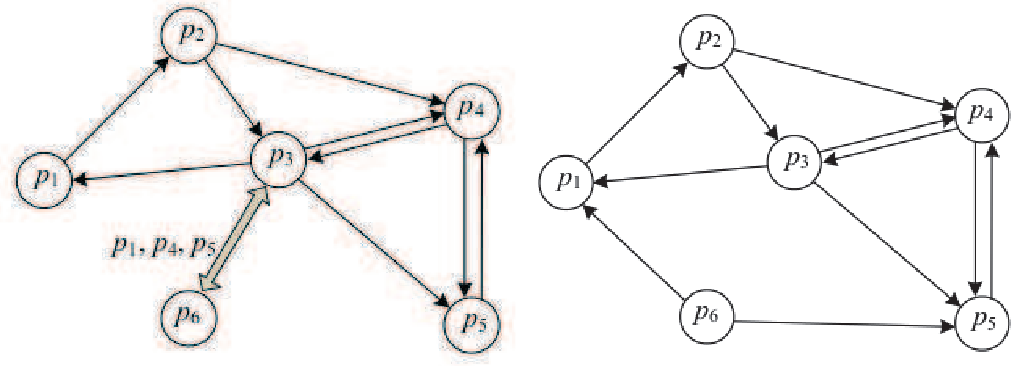
Topology management Although desirable, it is typically very difficult to maintain a global state of the system on every node. Indeed, this raises synchronization problems especially when the set of participating nodes changes quickly [33, 129, 134].

Gossip-based topology management protocols tackle this challenge by assuming that each peer knows only a limited subset of other nodes (neighbors), constituting its partial and local view of the complete network. Then, through continuously exchanging their neighbors and refreshing their local views, peers can self-organize into different overlay networks forming the basis for various applications.

More precisely, when gossiping, each peer picks a peer from its local view with the PeerSelection function to gossip with. The type of information exchanged between peers is the list or a subset of nodes from their local views, selected by the DataExchange function. After a gossip, the DataProcessing function updates the local view by incorporating received memberships if necessary.

Figure 2.2 depicts a joining node operation in the network. The node $p6$ starts by gossiping with a node $p3$ who is already in the network. Such a bootstrap operation can be discovered in various ways, including broadcasting in the local network, using a designated multicast group, etc. $p3$ exchanges its neighbors list ($p1$, $p4$, $p5$) with $p6$. Then $p6$ can select $p1$ and $p5$ for its own local view according to the targeted application. Once member of the network, $p6$ can further gossip with its neighbors to refine its view.

The targeted application drives the chosen network overlay. Typically, two main overlay are considered: (i) a random topology ensuring interesting properties such as small diameter or small clustering coefficient [65], and (ii) forming a structured topology based on specific criteria such as interests [67, 86].



(a) Gossip (b) Update partial view

Figure 2.2: *Example of gossip-based topology management [27].*

Information dissemination Gossip is analogous to an epidemic, where a virus plays the role of a piece of information, and infection plays the role of learning about the information (*e.g.* SIR model [60]). Figure 2.3 depicts how the information is disseminated through gossip among peers. Basically, at each cycle, the peers who have a news item gossip it to known peers. The number of targets a node gossips to a news item is called $fanout$, in the example depicted bellow, the fanout is 1. After a few cycles, all the peers in the network have received the information. It has been shown for instance, that with a $log(n)$ fanout, all nodes receive the message, n being the number of nodes in the system.

Related works

The scalability and the robustness of gossip protocols have inspired several works in the field of pub/sub such as Rappel [115], StAN [106], SpiderCast [50] or Tera [30]. These approaches rely on gossip protocols to identify similar users and to construct a separate overlay for each topic. When a node subscribes to a topic, it becomes a member of that topic overlay. Therefore, published events for that topic are only distributed among the associated subscriber nodes. However, nodes should join as many overlays as the number of topics they subscribe to. Thus, for large numbers of subscriptions, the node degree and overlay maintenance overhead grow and become an obstacle to scalability. To address this problem, SpiderCast bounds the number of connections on each node, independently to the number of their subscriptions.

Vitis [122], is a hybrid approach enabling rendezvous routings on unstructured overlays. Vitis also has a bounded node degree and relies on a gossip protocol to group nodes according to both their interests in similar topics and their event publication rate for various topics. However, the system is organized as a tree-like structure with rendezvous routing but the leaves in these trees are not single nodes but groups of similar nodes.

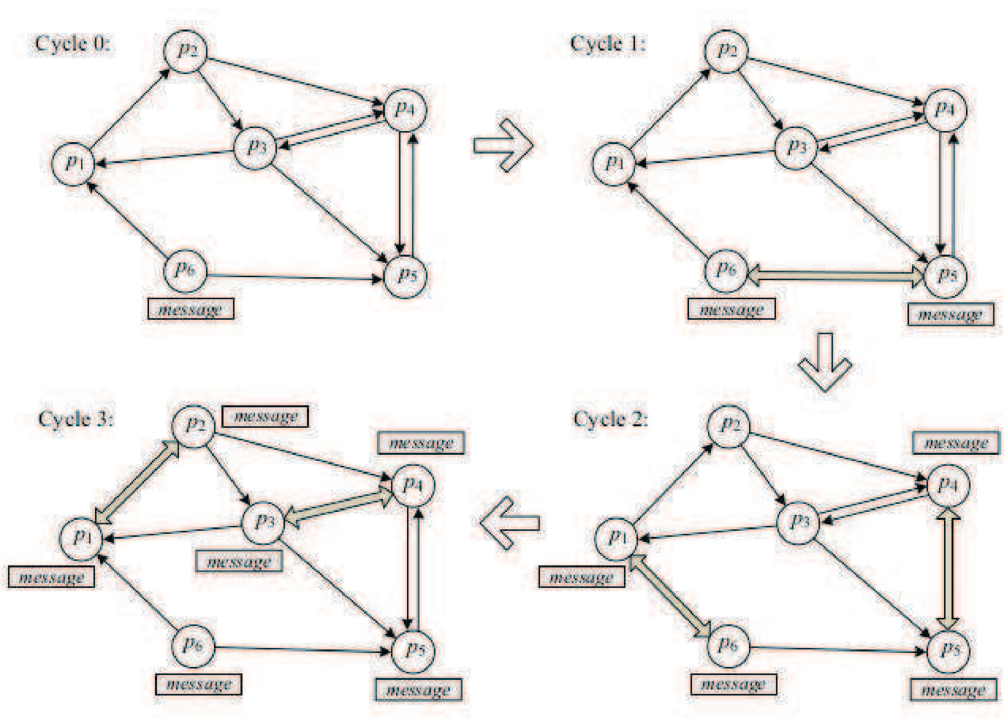


Figure 2.3: *Example of gossip-based information dissemination [27].*

Quasar [144], relies on a probabilistically model to deliver events to subscribers. In this solution, the node degree is also bounded and each node exchanges with its nearby neighbors an aggregated from of its subscriptions and subscriptions of close neighbors. This mechanism allows group members to install routing vectors in nearby overlay neighbors. Therefore, a gradient of group members for each topic emerges in the overlay. A node publishing an event for a specific group sends it through multiple random walks along the overlays. This event follows the routing vectors for finding the closest group members.

Analysis

Pub/sub systems rely on an explicit classification scheme: users are required to explicitly define their interests. This scheme is coarse grained and not adapted for a personalized feed of news where users can quickly change of interest according to the flow of news items. In addition, in such systems, a news item is delivered to all subscribers even if it is irrelevant.

2.2 News personalization

The goal of news personalization systems is to assist users in finding the information that matches their current interests. Personalization systems usually rely on recommendation systems. In general, a recommendation system (RS) accomplishes two tasks. The first is to collect data about users in order to understand their interests. The second is to provide relevant recommendations related to the current interests of the user. Compared to personalization in other fields which assumes that the underlying set of items is either static or the amount of item churn (insertions and deletions) is minimal, automatically exploiting user opinions and dynamically performing personalization on a huge number of items which undergoes churn every few minutes is a Dantean task. It is even more so with continuous streams of news items that need to be filtered and instantly delivered to users potentially changing their mind about what is relevant. In this section, we introduce the main challenges of a RS, the main techniques described in the literature, their evaluation and review some related works in both centralized and distributed architectures.

2.2.1 Challenges against news personalization

One of the main challenges underlying CF is **scalability**. This is particularly true for websites that enable users to generate content, a feature that is quickly becoming the norm. As a consequence, it is important that recommender algorithms scale well with the number of users and items. **Sparsity** is also an important issue of RS. Considering the large number of items, each user can only rate a small fraction of them. The recommender system must process optimally the data at its disposal. In addition, a RS must be able to provide recommendations for new users and to propose news items as soon as possible. The presence of new users with empty profiles and new items without ratings leads to the so-called **cold-start** issue. Lastly, RS and personalization raises a multitude of **privacy** challenges. As RSs rely on user activity to detect similarity patterns, they build and manipulate datasets gathering private and sensitive information about users and their consuming behavior.

2.2.2 Recommendation techniques

Different approaches have been proposed to build recommendation systems. While content-based recommenders use item descriptions to associate items with users (*e.g.* NewsWeeder [98], Krakatoa Chronicles [34], PersoNews [31], [21] or [72]), content-agnostic approaches such as Collaborative Filtering (CF) schemes are a better match for settings where content characterization is not always possible. CF is an appealing and the prevalent approach to provide users with recommendations on items [95, 135]. They are classified in two general categories, the memory-based and the model-based scheme. While the memory-based approaches, also known as neighborhood-based or k-nearest neighbor approaches, use directly the history of user activities to estimate the similarity between users (user-based approach) or item (item-based approach), the model-based approaches use the rating history to learn a model for prediction. Many models have been suggested for CF such as matrix factorization based on Singular Value Decomposition [96, 97, 116], graph-based methods [80], information theoretic approaches based on maximum entropy [100], or latent semantic analysis [79]. Hybrid recommenders [43], in turn, use different techniques together mainly to alleviate the drawbacks of one with the advantage to the others. In this work, we consider a user-based collaborative filtering scheme to provide personalization.

2.2.3 Evaluation of recommendation systems

Evaluation of recommendation system has always been challenging. The main reason is that it is hard to know the users' opinion about their recommendations. In addition, while some aspects of recommender systems like accuracy can be measured relatively easily, some more subjective goals like serendipity are more challenging to measure [78, 153]. RSs can be evaluated online or offline. The online evaluation consists of monitoring the reaction of users to their recommendations. This can only be done by content providers and the experiments are not repeatable. Most of the time, recommender systems are evaluated offline. The common method for offline evaluation is cross validation. In this method, the dataset is divided into two disjoint subsets: training set and test set. Predictions are made for the test set, and compared against the real values of the ratings. This approach allows for repeatable and comparable experiments.

Popular measures to estimate the total prediction error of an RS are the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE):

$$MAE = \frac{\sum_{r_{ui} \in R_{Test}} \|\widetilde{r}_{ui} - r_{ui}\|}{\|R_{Test}\|}$$

$$RMSE = \sqrt{\sum_{r_{ui} \in R_{Test}} (\widetilde{r}_{ui} - r_{ui}^2}\|R_{Test}\|$$

where r_{ui} is the rating of user u for item i, R_{Test} is the set of all ratings in the test set, and $\widetilde{r}_{ui}$ the estimation of the algorithm for r_{ui}. MAE and RMSE are proper to measures the global prediction error of an RS. However, if the recommendation function is to recommend a list of relevant items, *recall* and *precision* are the two most used metrics. Both measures are in $[0, 1]$. For an item, a recall of 1 means that all interested users have received the item. Yet, this measure does not account for spam since a trivial way to ensure a maximum recall is to send all news items to all users. This is why precision is required. A precision of 1 means that the news item has reached only the users that are interested in it. An important challenge in information retrieval is to provide a good trade-off between these two metrics. This is expressed by the F1-Score, defined as the harmonic mean of precision and recall [140].

$$Precision = \frac{|\{interested\ users\} \cap \{reached\ users\}|}{|\{reached\ users\}|}$$

$$Recall = \frac{|\{interested\ users\} \cap \{reached\ users\}|}{|\{interested\ users\}|}$$

$$F1 - Score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

2.2.4 Similarity metrics

Recommendation systems based on user-based collaborative filtering rely on similarity between users. In this section, we review some metrics proposed in the literature [93]. This list is not exhaustive, other measures exist (co-occurrence, etc).

Overlap

This simple similarity measure can be declined in different ways according to the application, and closely depends on how the users' profile is built and their content. In a news recommender context, where user profiles contain liked news items, this metric can be defined as the number of common items that two users u and v are interested in:

$$Overlap(u, v) = |Profile(u) \cap Profile(v)|$$

While simple, the lack of normalization of this metric may significantly favor users that have a large number of items in their profile. This drawback is partially fixed with cosine similarity presented bellow.

Cosine similarity

The cosine similarity metric has been extensively exploited in data mining [140]. This metric measures the cosine of the angle between two vector profiles and thus determines whether they are pointing in roughly the same direction. Given two vectors, $v1$ and $v2$, the cosine similarity, θ, is defined as

$$cos(v1, v2) = \frac{v1 \cdot v2}{\|v1\| \|v2\|}$$

Let I_u be the set of items in the profile of user u, the cosine between two users u and v is defined as follows:

$$cosineSimilarity = \frac{|Profile(u) \cap Profile(v)|}{\sqrt{|Profile(u) \times Profile(v)|}}$$

This similarity metric is normalized in $[O : 1]$ and can be viewed as the normalized overlap where the number of share items increased the similarity score between two users while non-shared interests decrease it.

Pearson correlation

In RSs using relative ratings, Pearson correlation and its variants (weighted Pearson correlation) are often preferred over the cosine similarity. Indeed, when users express their preferences through a range of values as in the 5-star rating system for instance, users rate the items from 1 to 5 stars but a user may rate from 3 to 5 with the same opinion on items than another user which rates from 1 to 3. The Pearson correlation lifts this drawback by considering the offset of each rating from the user's mean rating. It is defined as follows:

$$\rho_{u,v} = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - r_u)(r_{vi} - r_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - r_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - r_u)^2}}$$

where r_u is the mean rating of user u and r_{ui} is the rating of user u for item i.

Pearson correlation considers only the intersection of items rated by both users as the overlap similarity. As a result, a user may choose a neighbor with only very few items in common. To overcome this limitation, a significance weighting [] can be used. This is achieved by multiplying the Pearson correlation by a term reflecting the number of common items.

Multi-interest cosine similarity

Considering individual rating to select users having the most similar profiles may lead to take into account only the dominant interests, ignoring minor or emerging ones until they represent a large portion on the profile. To account for all interests of a user, a multi-interest similarity metric has been proposed in [32]. The idea behind this is to rate a set of user as a whole rather than each profile independently. This involves a balance between the extent to which nodes in the set share interests with a given node n, and how well the distribution of the interests in the set matches the one in n's profile.

Let I_u be the vector that represents the items in the profile of user u and $SetI_u(s)$ an item vector that represents the distribution of items in a set of nodes s with respect to the node u. Each value of this vector is computed as follows:

$$SetI_u(s)[i] = I_u[i] \times \sum_{v \in s} \frac{I_u[i]}{||||I_u||||}$$

The rationale behind this metric is to represent the distribution of the interests of u without having to explicitly define such interests while normalizing the contribution of each user of the set to favor specific interests. Items that are not present in the profile of u are naturally discarded. Then, user u computes a score for a set of nodes as follows:

$$SetScore_u(s) = (SetI_u(s) \cdot I_u) \times cosine(I_u, SetI_u(s))^b$$

This formula accounts for both the distribution of items in s (first part) and how well this distribution matches the one in u's profile (second part). Parameter b is the balances between the amount of shared interests and a fair distribution that does not favor any item. If $b = 0$ (no cosine impact), the distribution in not considered and the similarity metric therefore the same as the one obtained from the individual ratings.

2.2.5 Centralized approaches

The choice of the approach and the design of the underlying centralized architecture for personalization is mainly driven to address scalability. While dimension reduction and algorithmic optimizations [58, 64, 70, 71] partially tackle the problem, they do not remove the need to increase computational resources when the number of users and items increases [41, 52, 114].

Due to their high popularity, more and more personalization systems [11], open-source frameworks [4] or libraries [9] are proposed. The new standard in centralized approaches leverage massive parallelization through map-reduce jobs [56] on elastic cloud architectures. To address news personalization, only a handful of companies can afford the cost of scaling collaborative filtering with centralized architectures hosted in the cloud. Even with such massive parallelization, Google news personalization [52] confesses its inherent difficulty in coping with dynamics. Recommendations are not user centric but based on large clusters of users. Indeed, MinHash probabilistic clustering algorithm is leveraged to reduce the user base into clusters of similar users. Recommendations are then provided to users according to the popularity of news items in their clusters. Clusters of users are not updated on the fly but periodically (several hours) computed off-line. This latency can be an obstacle to relay quickly emerging news items.

In addition, the cost of a collaborative filtering system is high both in terms of hardware, whether owned or rented, and energy consumption. Data centers consume an enormous amount of power and energy requirements are responsible for a large fraction of their total cost of ownership and operation [49, 74, 108].

2.2.6 Decentralized approaches

Research on decentralized recommender systems is still modest despite their clear scalability advantage.

Tribler [150] is a decentralized content discovery and content recommendation implemented on top of the BitTorrent protocol. Tribler relies on a gossip protocol to form the set of similar users that should be considered to compute recommendations. This neighborhood is computed according to the cosine similarity metric using the vector profile of users. Vector profiles contain items where an entry for an item is 1 if the user has downloaded it, 0 otherwise. A user-based collaborative filtering approach is used to compute a score for each item, being consequently user to provide recommendations to user.

PocketLens [109] is a recommender algorithm developed by GroupeLens research group. PocketLens can rely on different architecture including fully decentralized ones, to form neighborhood in term of interests A cosine similarity metric is used to estimate the similarity between users. Then, an item-based algorithm is applied on the ratings gathered from the neighborhood to provide the recommendations.

In [82], the authors designed a decentralized matrix factorization approach based on gradient descent for predicting future user ratings in a user-generated content systems. Their experiments on the Netflix dataset and on a prototype implemented as a Facebook application show that this algorithm is competitive with centralized solutions.

Some approaches leverage the explicit social structure of the network to achieve selective dissemination. GoDisco [55] disseminates information through gossip in an explicit social network enriched with bridges between communities. Yet, it relies on explicit interest classification and node categorization, both requiring an upfront analysis of content. Similarly, the Friendship-Interests Propagation model [148] leverages the homophily [1] of explicit social networks to filter messages.

Similar to our solutions, several approaches have leveraged the power of gossip protocols to build P2P network overlays and to assign each user with a neighborhood of similar peers. In these approaches, each user computes a network of social acquaintances by gossiping among each other and computing the proximity between profiles. Every user maintains her own social network but locally stores a limited subset of profiles, typically those of the most similar users.

Authors in [94] use this gossip scheme to propose distributed user-based collaborative filtering system addressing specifically sparse data. They use a modified Pearson correlation metric to assign each user with a neighborhood of similar peers. Then, each user locally runs the random walk algorithm on her neighborhood, and computes her recommendations.

GEOLOGY [46], is a fully-decentralized modular framework for recommending geolocations using the same gossip scheme to from neighborhood. They explore

[1]People with similar interest tend to connect to each other and people of similar interest are more likely to be friends.

different similarity metrics to evaluate the semantic proximity between users and proposed a technique mixing compaction, Adamic/Adar similarity metric and sampling.

GOSSPLE [32] is a fully decentralized anonymous collaborative network using this gossip scheme to produce query expansion mechanism. Users continuously gossip digest of their profiles encoded in Bloom filters and locally compute a personalized view of the network. An extended version of the cosine similarity (presented in Section 2.2.4) is used to cover the various interests of the user without any explicit support such as explicit social links or ontology.

Similar scheme is used in P3Q and P4Q [28, 29], fully decentralized approaches to personalized top-k and query processing, respectively. However, the maintenance of the overlay network in P3Q is performed in a lazy mode to avoid overloading the network. Users exchange with a low frequency a digest of their profile encoded in Bloom filters to first estimate the proximity between profiles. Then, when a digest appear to be significantly similar, the whole profile is exchanged. P4Q, in turn, uses an greedy and biased mode for the gossip protocol. Every query is first computed locally based on the set of stored profiles, providing an immediate result to the user. The query is then gossiped, first to the closest acquaintances and further away according to social proximity, iteratively refining the results.

Analysis

Most of news personalization systems are centralized and require massive architecture. To cope with scalability, they are not user centric but cluster-based, and rely on off-line clustering operations. Decentralized recommender systems, in turn, are applied to much less dynamic contexts than instant news and do not integrate mechanism to efficiently filter irrelevant news items.

Chapter 3

Decentralizing news recommender

This chapter presents WhatsUp, *a collaborative filtering system for disseminating news items in a large-scale dynamic setting with no central authority.* WhatsUp *constructs an implicit social network based on user profiles that express the opinions of users about the news items they receive (like-dislike). Users with similar tastes are clustered using a similarity metric reflecting long-standing and emerging (dis)interests. News items are disseminated through a novel heterogeneous gossip protocol that (1) biases the orientation of its targets towards those with similar interests, and (2) amplifies dissemination based on the level of interest in every news item.*

We report on an extensive evaluation of WhatsUp *through (a) simulations, (b) a ModelNet emulation on a cluster, and (c) a PlanetLab deployment based on real datasets. We show that* WhatsUp *outperforms various alternatives in terms of accurate and complete delivery of relevant news items while preserving the fundamental advantages of standard gossip: namely, simplicity of deployment and robustness.*

3.1 Introduction

The stream of news items we are exposed to is huge and keeps growing exponentially. This calls for automatic techniques to filter the right content for every one, alleviating the need to spend a substantial amount of time browsing information online. Explicit subscription-based approaches (*e.g.* RSS, pub/sub, online social networks) are not always relevant in this context: they either filter too much or not enough. *Personalized news recommender systems*, based on so-called social or collaborative filtering (CF) [135], are much more appropriate for they operate in a dynamic and fine-grained manner to automate the celebrated *word-of-mouth* pattern by which people recommend useful items to each other. However, CF approaches require the maintenance of huge amounts of information as well as significant computation resources, especially in the context of continuous streams of news items that must be instantly delivered to users that potentially change interests over time.

The motivation of this work is to determine whether a CF instant news system is feasible in a completely decentralized manner. Intuitively, a P2P approach is attractive because it naturally scales and circumvents a central entity that controls all user profiles potentially exploiting them for commercial purposes. Yet, the absence of a central authority with global knowledge makes the filtering very challenging and calls for CF schemes that need to cope with partial and dynamic interest profiles.

We present, in this chapter, WhatsUp: the first decentralized instant news recommender system. WhatsUp consists of a simple user interface and two distributed protocols: Wup and Beep (Figure 3.1), which are key in providing an *implicit publish-subscribe* abstraction. They enable users to receive published items without having to specify explicit subscription filters. The user interface captures the opinions of users on the news items they receive through a simple *like/dislike* button.[1] A user *profile* collects the resulting implicit interests in a vector associating news items with user opinions. This provides the driving information for the operation of Wup and Beep.

Wup maintains a dynamic implicit social network, a directed graph linking nodes (reflecting users) with similar interests. Wup periodically samples the network by gossiping profiles and connects similar users by mixing randomization to seek completeness (or *recall*), and similarities to seek accuracy (or *precision*). The similarity metric we consider accounts for the ever-changing interests of users and prevents the formation of isolated islands of interest.

News items are disseminated using Beep (Biased EpidEmic Protocol), a novel heterogeneous epidemic protocol obeying the *explore-and-exploit* principle. The protocol biases dissemination towards nodes that are likely to have similar tastes (*exploit*), while introducing enough randomness and serendipity (ability of making fortunate discoveries while looking for something unrelated) to tolerate the inherent

[1] The axiom underlying WhatsUp, as for any CF scheme [135], is that users who have exhibited similar tastes in the past are likely to be interested in the same news items in the future.

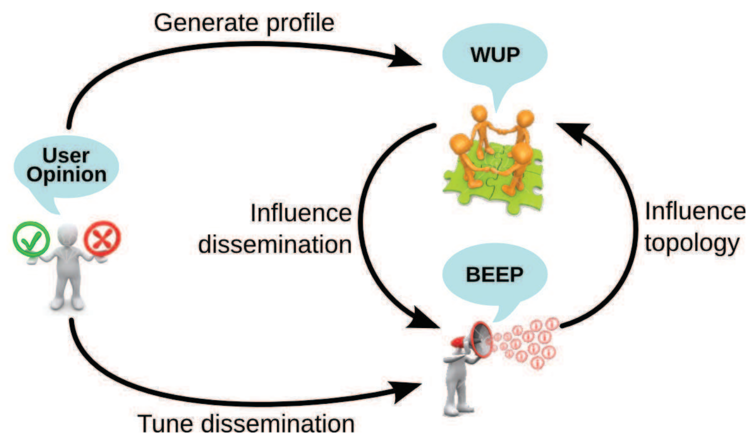


Figure 3.1: *Interactions between (1) user opinion; (2)* Wup*: implicit social network; (3)* Beep*: news dissemination protocol.*

unreliability of the underlying network as well as to prevent interesting news items from being isolated within specific parts of the network (*explore*). If a user likes a news item, Beep forwards it along the (implicit) social-network topology constructed using Wup. Otherwise, Beep gives the item a chance to visit other parts of the network. The news-dissemination process generates a wave of profile updates, in turn potentially impacting the Wup network topology. Unlike classical gossip protocols [60], which aim at delivering news items to all users, Beep targets specific subsets of users determined dynamically for each news item. To our knowledge, it is the first gossip protocol to provide *heterogeneity* along multiple dimensions: *amplification* and *orientation*. *Amplification* tunes the number of times a node gossips a news item (its *fanout*). This acts as a social filter based on the opinions of the users exposed to news items. *Orientation* biases the choice of gossip targets towards users with similar tastes.

We fully implemented WhatsUp and we extensively evaluated it both by simulation and by deploying it over a cluster as well as on PlanetLab. Specifically, our results compare the performance of WhatsUp against various alternatives, including social cascades, a traditional topic-based pub/sub system, as well as distributed CF schemes.

☆ Contributions

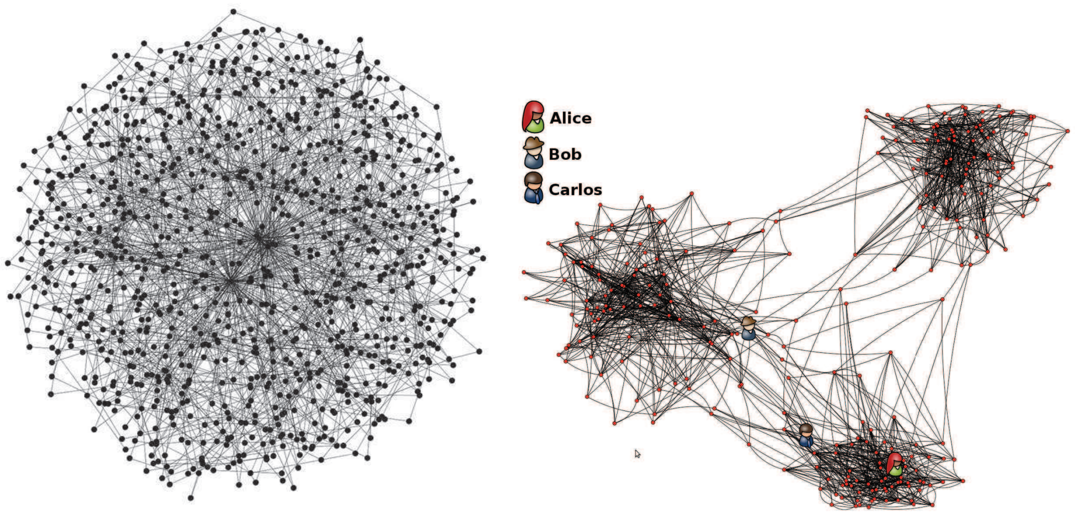
To summarize, this chapter presents two contributions, each, we believe, interesting in its own right: a clustering protocol, Wup, integrating a proximity metric, and Beep, a heterogeneous dissemination protocol. The integration of these protocols within the same coherent system, of which we provide an implementation and extensive evaluation can also be viewed as an actual contribution.

This chapter is organized as follows, Section 3.2 and Section 3.3 present respectively the two core components of WhatsUp, Wup, a clustering protocol and Beep, a heterogeneous dissemination protocol. Section 5.5 details our implementation while Section 4.7 provides the experimental setup of WhatsUp's evaluation. Section 4.8 presents our extensive evaluation and Section 4.2 gives an overview of some related works before concluding in Section 4.9

3.2 WUP

Wup builds and maintains an implicit social network and is itself based on two gossip protocols. The lower-layer random-peer-sampling (rps) protocol [87] ensures connectivity by building and maintaining a continuously changing random topology. The upper-layer clustering protocol [143] uses this overlay to provide nodes with the most similar candidates to form their Wup social networks.

At each node n, each protocol maintains a *view*, a data structure containing references to other nodes: the rps neighbors and the Wup neighbors. Each entry in each view is associated with a node and contains *(i)* its IP address, *(ii)* its node ID, *(iii)* its *profile* (as defined in Section 3.2.2), as well as *(iv)* a timestamp specifying when the information in the entry was generated by the associated node. Periodically, each protocol selects the entry in its view with the oldest timestamp [87] and sends it a message containing its profile with half of its view in the case of the rps (typical parameter in such protocols), or its entire view for Wup (the view sizes and frequencies of each protocol are given in Section 4.7).



(a) Random overlay (b) Wup overlay

Figure 3.2: *Overlay provided by the lower-layer random-peer-sampling protocol (left) and by the upper-layer clustering protocol (right).*

In the rps, the receiving node renews its view by keeping a random sample of the union of its own view and the received one. The union of the rps views represents a continuously changing random graph [87]. In Wup, instead, the receiving node selects the nodes from the union of its own and the received views whose profiles are closest to its own according to a similarity metric. This metric is an asymmetric variation of the well-known cosine similarity [136]: it seeks to maximize the number of items that were liked in both profiles being compared. It also strives to minimize spam by discouraging a node, n, with profile P_n, from selecting a neighbor, c, with profile P_c, that explicitly dislikes the items that n likes. We achieve this by dividing the number of liked items in common between the two profiles by the number of items liked by n on which c expressed an opinion. We define $sub(P_n, P_c)$ as the subset of the scores in P_n associated with the items that are present in P_c. By further dividing by the number of items liked by c (as in cosine similarity), we then favor neighbors that have more restrictive tastes. The asymmetric structure of this metric is particularly suited to push dissemination (*i.e.* users choose the next hops of news items but have no control on who sends items to them) and improves cold start with respect to cosine similarity as explained in Section 3.6.1.

$$Similarity(n, c) = \frac{sub(P_n, P_c) \cdot P_c}{\|sub(P_n, P_c)\| \, \|P_c\|}$$

3.2.1 News item

WhatsUp disseminates news items. In addition to the title, a news item consists of a short description as well as of a link to further information. A node that generates an item, its source, associates it with a timestamp indicating its creation time and a dislike-counter field initialized to zero that sums the number of dislikes obtained by the item. The identifier of the news item consists of an 8-byte hash of its content and is not transmitted. Rather, it is computed by nodes when they receive the item.

3.2.2 Profiles

Wup records information about interest for items in profile data structures. A profile is a set of triplets: identifier, timestamp, and score; $P \in \{< id, t, s > | id \in N, t \in T, s \in [0, 1]\}$. Identifier and timestamp are defined as above, and each profile contains only a single entry for a given identifier. The score, instead, represents the level of interest for an item: 1 meaning interesting, and 0 not interesting.

Wup associates each node with a profile, the *user profile* ($\tilde{P}$), which contains information about the node's own interests. The scores associated with this profile are integer values (like-dislike). To disseminate news items, nodes employ an additional profile structure, the *item profile* (P^N). Unlike a user profile, the item profile is associated with a news item. Its score values are real numbers and are obtained through the aggregation of the profiles of the users that liked the item along its dissemination path. As a result, two copies of the same item along two different paths will have different profiles. This causes an item profile to reflect the interests of the portion of the network it has traversed. The item profile can also be viewed as a community profile expressing the interests of an implicit social network of nodes.

3.2.3 Updating profiles

Updating user profiles ($\tilde{P}$) A node updates its profile whenever it expresses its opinion on a news item either by clicking the *like* or the *dislike* button (line 5 or 42 in Algorithm 1), or when generating a new item (line 37). In either case, the node inserts a new tuple containing the news item's identifier, its timestamp, and a score value of 1 if it liked the item and 0 otherwise.

Updating item profiles (P^N) An item profile records the interests of the users who like an item by aggregating their profiles along its path. Let R be an item that a node, n, likes. When n receives R for the first time, it first updates its own user profile as described above. Then, it iterates through all its tuples (line 3). Let *id* be the identifier of one such tuple. Node n checks if R's item profile already contains a tuple for *id* (`addToNewsProfile` function). If so (line 20), n replaces the existing tuple's score with the average between the scores in the two profiles. In this case, the score of the current user profile has the same weight as the previous score which can relies on multiple users. This causes a personalization of the item profile according to the current user's interests. If the tuple for id is not already present, the node inserts its own tuple into R's profile (line 22). When a new item is generated (function `generateNewsItem` in Algorithm 1), the source initializes the corresponding item profile by integrating its own user profile (lines 15).

3.2.4 Initialization

A node, n, that is joining the system for the first time (*cold start*) contacts a random node, and inherits its RPS and WUP views. It then builds a fresh profile by selecting and rating the 3 most popular news items from the profiles of the nodes in its the selected RPS view. This process results in a profile and in a WUP view that are very unlikely to match n's interests. However, it provides n with a way to enter the WUP social network. Because the WUP metric takes into account the size of user profiles, nodes with very small profiles containing popular items such as joining nodes are more likely to be part of the WUP views of other nodes and quickly receive additional news items. This allows them to fill their profiles with more relevant content, thereby acquiring closer neighbors.

3.2.5 *Profile window*

The information stream is continuously evolving. In order to take into account only the current interests of users and to dynamically connect similar users, all profiles are cleaned of old items. Specifically, each node periodically purges its user profile of all

Algorithm 1: WUP: receiving / generating an item.

```
on receive (item $< id^N, t^N >$, profile $P^N$, dislike counter $d^N$) do
    if iLike($id^N$) then
        for all $< id, t, s > \in \tilde{P}$
            addToNewsProfile($< id, t, s >$, $P^N$)
        add $< id^N, t^N, 1 >$ to $\tilde{P}$
    else
        add $< id^N, t^N, 0 >$ to $\tilde{P}$
    for all $< id, t, s > \in P^N$
        if $t$ older than profile window then
            remove $< id, t, s >$ from $P^N$
    BEEP.forward(($< id^N, t^N >$, $P^N$, $d^N$))
function generateNewsItem(item $id^N$)
    add $< id^N, t^N, 1 >$ to $\tilde{P}$
    $P^N \leftarrow \emptyset$;    $d^N \leftarrow 0$;    $t^N \leftarrow currentTime$
    for all $< id, t, s > \in \tilde{P}$
        addToNewsProfile($< id, t, s >$, $P^N$)
    BEEP.forward(($< id^N, t^N >$, $P^N$, $d^N$))
function addToNewsProfile($< id^N, t, s >$, $P^N$)
    if $\exists s^n | < id^N, t, * > \in P^N$ then
        $s^n \leftarrow \frac{s^n + s}{2}$
    else
        $P^N \leftarrow < id^N, t, s >$
```

the tuples whose timestamps are older than a profile window. Similarly, nodes purge item profiles of non-recent items before forwarding items to BEEP for dissemination (lines 8 to 10). The value of this profile window defines the reactivity of the system with respect to user interests as discussed in Section 3.5.4.

It is important to note that the profile window also causes inactive users who have not provided ratings during the current window to have empty profiles, thus being considered as new nodes. Yet, as in the case of initialization, the WUP metric allows these users to reintegrate quickly as soon as they connect and resume receiving news items.

3.3 BEEP

BEEP is a novel gossip-based dissemination protocol embodying two mechanisms: *orientation* and *amplification*, both triggered by the opinions of users on news items. Orientation leverages the information provided by WUP to direct news items towards the nodes that are most likely to be interested in them. Amplification varies the number of dissemination targets according to the probability of performing a useful forwarding action. Orientation and amplification make BEEP the first user-driven gossip protocol to provide heterogeneity in the choice as well as in the number of

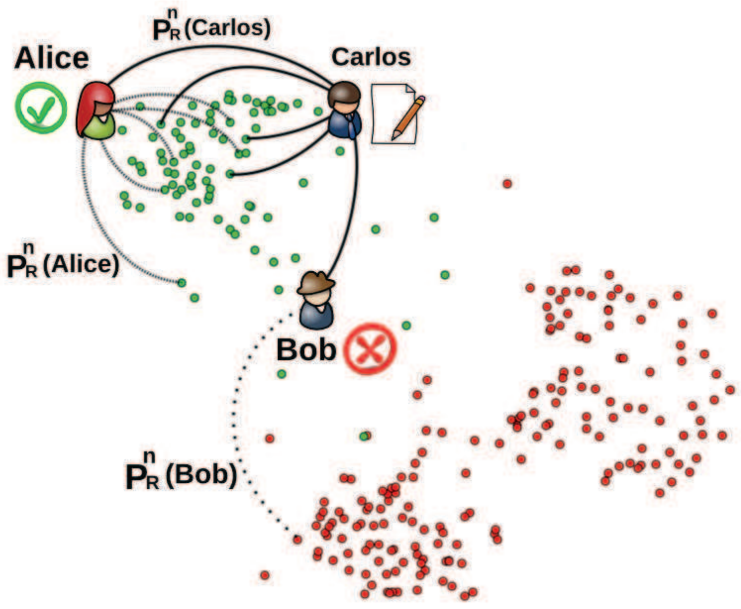


Figure 3.3: *Orientation and amplification mechanisms of Beep.*

dissemination targets, achieving differentiated delivery. Beep follows the well-known SIR (Susceptible, Infected, Removed) [60] model. A node receiving a news item for the first time updates the item's profile as described in Section 3.2.3. Then, it forwards the item to fanout (f) other nodes chosen according to its opinion on the item, as described in the following. A node receiving an item it has already received simply drops it.

3.3.1 Forwarding a disliked item

With reference to Algorithm 2 and Figure 3.3, consider Bob, who does not like item R sent by Carlos. Beep first verifies if the dislike-counter field of the item has already reached the prescribed TTL (line 25). If it has, it drops the item. Otherwise it increments its value, and achieves orientation by identifying the node from Bob's RPS view whose user profile is closest to the item's profile (line 27) and forwards the item to it (line 53). The item profile allows Beep's orientation mechanism to identify a target that is reasonably close to someone who liked the item, even if its topic falls outside Bob's interests. The use of a fanout of 1, instead, accounts for unexpected interests and addresses serendipity by giving news items the chance to visit portions of the overlay where more interested nodes are present. At the same time, it also prevents non-interesting items from invading too many users.

Algorithm 2: BEEP: forwarding a news item.

```
23 function forward(($< id^N, t^N >$, profile $P^N$, dislike counter $d^N$))
24     if ¬ iLike($id^N$) then
25         if $d^N <$ TTL then
26             $d^N \leftarrow d^N + 1$
27             $N \leftarrow$ selectMostSimilarNode($P^N$, RPS )
28         else
29             $N \leftarrow \emptyset$
30     else
31         $N \leftarrow$ selectRandomSubsetOfSize(WUP, $f_{\text{LIKE}}$ )
32     if $N \neq \emptyset$ then
33         for all $n \in N$
34             send $< id^N, t^N >$ with associated $P^N$ and $d^N$ to $n$
```

3.3.2 Forwarding a liked item

Consider now Alice (Figure 3.3), who instead finds item R interesting. BEEP achieves orientation by selecting dissemination targets from her social network (WUP view). Unlike the profiles in the RPS view, those in the WUP view are relatively similar to each other. However, to avoid the formation of too clustered a topology by selecting only the closest neighbors, BEEP selects its targets randomly from the WUP view (line 31 in Algorithm 2). Moreover, since the targets' interests are expected to be similar to those of the node, BEEP amplifies R by selecting a relatively large subset of f_{LIKE} (*like fanout*) nodes instead of only one node, thus giving R the ability to reach more interested nodes.

3.4 Implementation

WHATSUP is implemented in Java as a simple-to-deploy Web application. Its architecture consists of two main components on each node: the Web user interface and the application server (Figure 3.4), both locally hosted by each node and enable the corresponding user to interact with the rest of the system.

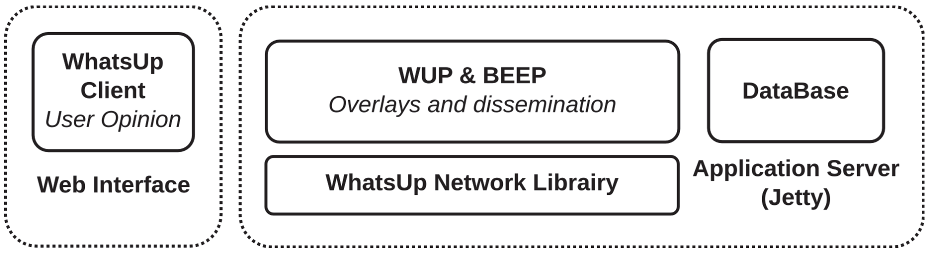
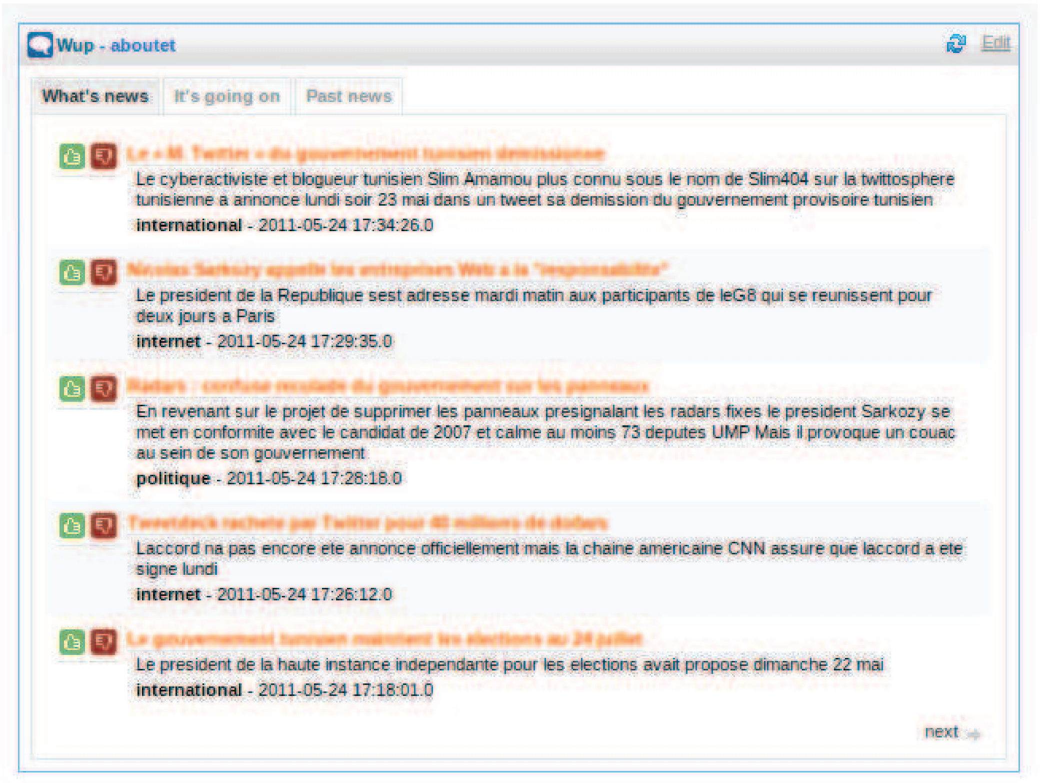


Figure 3.4: *Architecture of* WHATSUP*: each node locally hosts the application server part providing a Web interface for the user to interact with the system.*


Wup - aboutet

Edit

What's news | It's going on | Past news

Le « M. Twitter » du gouvernement tunisien demissionne
Le cyberactiviste et blogueur tunisien Slim Amamou plus connu sous le nom de Slim404 sur la twittosphere tunisienne a annonce lundi soir 23 mai dans un tweet sa demission du gouvernement provisoire tunisien
international - 2011-05-24 17:34:26.0

Nicolas Sarkozy appelle les entreprises Web a la "responsabilite"
Le president de la Republique sest adresse mardi matin aux participants de leG8 qui se reunissent pour deux jours a Paris
internet - 2011-05-24 17:29:35.0

Radars : confuse reculade du gouvernement sur les panneaux
En revenant sur le projet de supprimer les panneaux presignalant les radars fixes le president Sarkozy se met en conformite avec le candidat de 2007 et calme au moins 73 deputes UMP Mais il provoque un couac au sein de son gouvernement
politique - 2011-05-24 17:28:18.0

Tweetdeck rachete par Twitter pour 40 millions de dollars
Laccord na pas encore ete annonce officiellement mais la chaine americaine CNN assure que laccord a ete signe lundi
internet - 2011-05-24 17:26:12.0

Le gouvernement tunisien maintient les elections au 24 juillet
Le president de la haute instance independante pour les elections avait propose dimanche 22 mai
international - 2011-05-24 17:18:01.0

next

Figure 3.5: *Information stream produces by* WhatsUp*'s user web interface.*

Users may use multiple devices to access their news feeds. This requires mechanisms to synchronize their personal data between devices. These are however outside the scope of our current work. Users only require a browser to interact with the system. The WhatsUp user interface is, in fact, a fully dynamic Web widget that can be integrated in both dashboards and Web pages. The widget displays the news received and ranked by the application server and collects user opinions in order to trigger the dissemination. In addition to a title and a Web link, the interface associates each news item with a short description, enabling the user to express her opinion without needing to read the full article. It also allows the user to retrieve past news items and opinions as well as easily generate news items. In addition, a tool-bar integrated into the browser allows users to share news items obtained from other websites.

The application server continuously updates the user interface with a stream of news items received from other nodes. To make this possible, it combines the implementations of Wup and Beep with a lightweight local database containing information on the user's profile. The underlying network library, designed for the management of gossip-based overlays, also provides support for peers operating behind NATs [75].

3.5 Experimental setup

In this section, we provide the experimental setup of WhatsUp's evaluation: the workloads, the competitors we compared against, WhatsUp's parameters, and the evaluation metrics we use to assess the performance of WhatsUp.

3.5.1 Datasets

We evaluate WhatsUp using several datasets: *(i)* a 3180-user synthetic trace derived from Arxiv, *(ii)* a Digg dataset crawled in late 2010, and *(iii)* a survey conducted in our lab providing a real set of WhatsUp users. Table 3.1 summarizes the figures of the workloads used in our evaluations.

Name	Number of users	Number of news
Synthetic	3180 Arvix Users	2000
Digg	750	2500
WhatsUp Survey	480	1000

Table 3.1: Summary of the workloads

Synthetic dataset To validate WhatsUp without the artifacts of real datasets, we identified distinct groups among the 5242 users in the Arxiv dataset (covering scientific collaborations between authors [131]) using a community-detection algorithm [113]. This allows us to deal with clearly defined communities of interest, thus enabling the evaluation of WhatsUp's performance in a clearly identified topology. The resulting dataset contains 21 communities ranging in size from 31 to 1036, for a total of 3703 users. For each community, we use a random subset of nodes as sources to disseminate 120 news items (for a total of about 2500).

Digg dataset Digg is a centralized social-news website designed to help users discover and share content. It disseminates news items along the edges of an explicit social network (*i.e.* cascading). Relying on explicitly declared friends, as in Digg, is known to limit the content that can be received [147] by substantially influencing decision making [88]. Basically, users are only exposed to the content forwarded by their friends, while other items may be of interest to them. To remove this bias, we extracted for each user, u, the categories of the news items she generates. We then defined user u's interests by including all the news items associated with these categories. We collected traces from Digg over 3 weeks in 2010. The resulting dataset consists of 750 users and 2500 news from 40 categories.

Dataset from a WhatsUp user survey We also conducted a survey on 200 news items involving 120 colleagues and relatives. We selected news randomly from a set of RSS feeds illustrating various topics (culture, politics, people, sports, ...). We exposed this list to our test users and gathered their reactions (like/dislike) to each news item. This provided us with a small but *real* dataset of WhatsUp users exposed to exactly the same news items. To scale our system, we generated 4 instances of each user and news item in the experiments. Yet, the resulting bias affects both WhatsUp and the state-of-the-art solutions we compare against.

3.5.2 WhatsUp Competitors

In order to demonstrate the effectiveness of WhatsUp, we evaluate it against the following alternatives:

Explicit cascading Cascading is a dissemination approach followed by several social applications, e.g., Twitter, Digg. Whenever a node *likes* (tweets in Twitter and diggs in Digg) a news item, it forwards it to all of its explicit social neighbors. We compare WhatsUp against cascading in the only dataset for which an explicit social network is available, namely Digg.

Complete explicit pub/sub WhatsUp can be seen as an *implicit publish/subscribe* (pub/sub) system turning interests into implicit subscriptions. Typically, pub/sub systems are explicit: users explicitly choose specific topics [16, 50, 66, 138]. Here, we compare WhatsUp against C-Pub/Sub, a centralized topic-based pub/sub system achieving complete dissemination. C-Pub/Sub guarantees that all the nodes subscribed to a topic receive all the associated items. C-Pub/Sub is also ideal in terms of message complexity as it disseminates news items along trees that span all and only their subscribers. For the sake of our comparison, we extract explicit topics from keywords associated with the RSS feeds in our survey. Then we subscribe a user to a topic if she likes at least one item associated with that topic.

Decentralized collaborative filtering In a decentralized CF scheme based on nearest-neighbor technique, when a node receives a news item it likes, it forwards it to its k closest neighbors according to some similarity metric. We implemented two versions of this scheme: one relying on the same metric as WhatsUp (CF-Wup) and one relying on cosine similarity [136] (CF-Cos). While it is decentralized, this scheme does not benefit from the orientation and amplification mechanisms provided by Beep. More specifically, it takes no action when a node does not like a news item.

Centralized version of WhatsUp We also compare WhatsUp with a centralized system (C-WhatsUp) gathering the *global knowledge* of all the profiles of its users and news items. C-WhatsUp leverages this global information (vs a restricted sample of the network) to boost precision using complete search. When a user likes a news item, the server delivers it to the f_{LIKE} closest users according to the cosine similarity metric. In addition, it also provides the item to the f_{LIKE} users with the highest correlation with the *item's profile*. When a user does not like an item, the server presents it to the f_{DISLIKE} nodes whose profiles are most similar to the item's profile (up to TTL times).

Centralized collaborative filtering Moreover, we compare WUP against centralized CF schemes. We considered a user-based CF system using a cosine similarity metric and an item-based CF one based on the slope-one predictor [101] provided by Mahout [9], an open-source machine-learning Apache library. We also confront WHATSUP with a CF scheme using the MinHash probabilistic clustering algorithm [52] dividing the user base into clusters of similar users available here [8]. We train each CF scheme by providing it, for each news item, with the opinions of the n_{START} users that are closest to item's source.

3.5.3 Evaluation metrics

We consider two types of metrics in our evaluation. User metrics measure the quality of WHATSUP's dissemination and its ability to filter content. They are important for users to decide whether to adopt WHATSUP as a system. In contrast, system metrics are transparent to users but are crucial to assessing the effectiveness of our solution.

User metrics We evaluated WHATSUP along the traditional metrics used in information-retrieval systems: *recall* (*i.e.* completeness) and *precision* (*i.e.* accuracy). These metrics are presented in Section 2.2.3.

System metrics To evaluate the behavior of WHATSUP from a systems perspective, we first consider the network traffic it generates. For simulations, we compute the total number of *sent messages*. For our implementation, we instead measure the average consumed *bandwidth*. Throughout our evaluation, we examine results obtained over a wide range of fanout values by plotting the F1-Score against the fanout, and against the number of generated messages. The F1-Score for corresponding fanout values makes it possible to understand and compare the behavior of WHATSUP and its competitors under similar conditions. The F1-Score for corresponding numbers of messages, instead, gives a clearer picture about the trade-offs between recommendation quality and cost. Two different protocols operating at the same fanout, in fact, do not necessarily generate the same amount of traffic.

3.5.4 WhatsUp system parameters

The operation of WHATSUP is controlled by a number of system parameters. The first two parameters we consider are the WUP view size (WUP_{vs}) and the BEEP-I-like

Parameter	Description	value
RPS$_{vs}$	Size of the random sample	30
RPS$_f$	Frequency of gossip in the RPS	1h
WUP$_{vs}$	Size of the social network	$2f_{\text{LIKE}}$
Profile window	News item TTL	13 cycles
BEEP TTL	Dissemination TTL for dislike	4

Table 3.2: WhatsUp *parameters - on each node*

fanout (f_{LIKE}). Clearly, the former must be at least as large as the latter. As a node forwards a liked news item to random neighbors among its Wup view, a WUP$_{vs}$ close to f_{LIKE} boosts precision while a large WUP$_{vs}$ compared to f_{LIKE} increases recall. We set the value of WUP$_{vs}$ to the double of f_{LIKE} as experiments provide the best trade-off between precision and recall for these values.

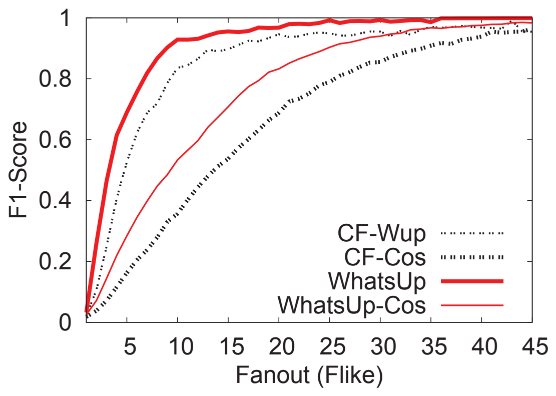
The third important parameter is the RPS view size. It directly impacts the potential of Wup to discover new nodes. We set its value to 30 to strike a balance between the need to discover information about nodes, the cost of gossiping, and the need to retain some randomness in the selection of Wup neighbors. Too large values would lead the Wup view to converge too fast, hampering the ability to address non-foreseeable interests (serendipity). Nonetheless, we verified that our protocol provides good performance with values between 20 and 40 in the considered traces.

The Beep TTL controls WhatsUp's *serendipity*, but it should not be too large in order not to hamper precision. We therefore set it to 4, and examine its impact in Section 3.6.2.

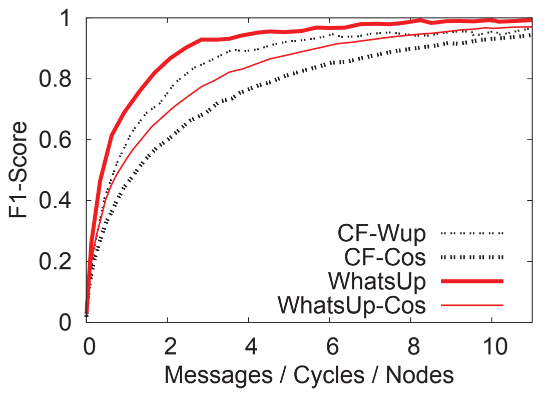
Finally, the size of the profile window determines WhatsUp's ability to adapt to dynamic and emerging interests of users. We set its value to 13 gossip cycles, corresponding to 1/5 of the experiment duration, according to an analysis of its influence on the F1-Score. A size between 1/5 and 2/5 of the whole period gives the best F1-Score, while smaller or larger values make WhatsUp either too dynamic or not enough. For practical reasons, our simulations use the duration of a gossip cycle as a time unit to represent the length of the profile window. Yet, the actual duration of a gossip cycle is important and determines the dynamic response of our system. We discuss this parameter and its impact when evaluating our deployment (Section 3.6.6).

3.6 Evaluation

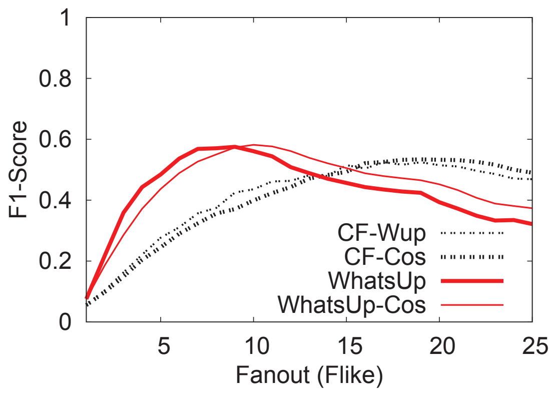
We carried out an extensive evaluation of WhatsUp by simulation and by deploying its implementation on PlanetLab and on a ModelNet-based [139] cluster. All parameters, based on observations on a wide range of experiments on all datasets, are summarized in Table 3.2. We present the results by highlighting each important feature of WhatsUp.



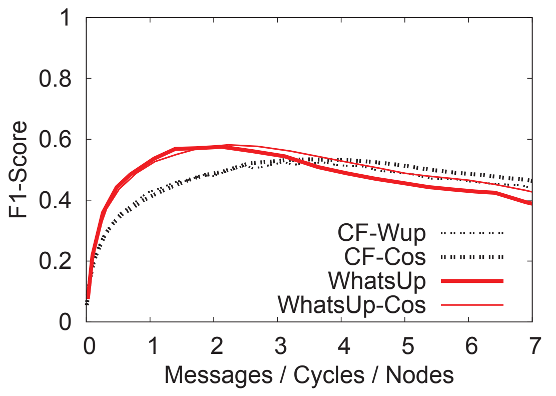
(a) Synthetic - fanout



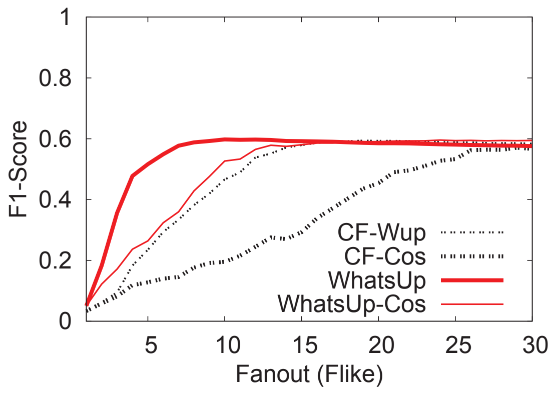
(b) Synthetic - message



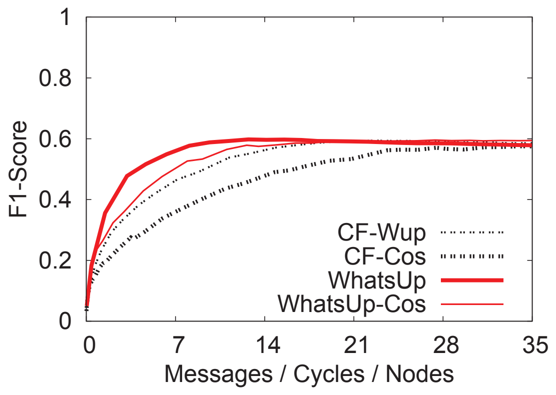
(c) Digg - fanout



(d) Digg - message



(e) Survey - fanout



(f) Survey - message

Figure 3.6: *F1-Score depending on the fanout and message cost*

Algorithm	Precision	Recall	F1-Score	Mess./User
Gossip ($f = 4$)	0.35	0.99	0.51	4.6k
CF-Cos ($k = 29$)	0.50	0.65	0.57	5.9k
CF-Wup ($k = 19$)	0.45	0.85	0.59	4.7k
WhatsUp-Cos ($f_{\text{LIKE}} = 24$)	0.51	0.72	0.60	4.3k
WhatsUp ($f_{\text{LIKE}} = 10$)	**0.47**	**0.83**	**0.60**	**2.4k**

Table 3.3: *Survey: best performance of each approach*

3.6.1 Similarity metric

We start by evaluating the effectiveness of the Wup metric. Figures 3.6a-3.6f compare two CF approaches and two versions of WhatsUp based, respectively, on cosine similarity (CF-Cos and WhatsUp-Cos) and our Wup metric (CF-Wup and WhatsUp). Our metric consistently outperforms cosine similarity in all datasets. Table 3.3 conveys the fact that it achieves this by improving recall over cosine similarity (by 30% for CF approaches and 15% for WhatsUp in the survey dataset with lower message cost in both cases). Moreover the relatively high precision of cosine similarity is partly an artifact of its low recall values resulting from highly clustered topologies. As a result, approaches using cosine similarity require a much larger fanout and message cost to provide the same quality of recommendation. The Wup metric generates instead topologies with a lower clustering coefficient by avoiding node concentration around hubs (an average clustering coefficient of 0.15 for Wup metric compared to 0.40 for cosine similarity in the survey dataset). In addition, the Wup metric avoids fragmenting the topology into several disconnected parts. Figure 3.7 shows the fraction of nodes that belong to the largest strongly connected component (LSCC) with increasing fanout values. Once all users are part of the same connected component, news items can be spread through any user and are not restricted to a subpart of the network. This corresponds to the plateaus in the F1-Score values visible in Figure 3.6e. The Wup metric reaches this state with fanout values around 10 both in CF-Wup and WhatsUp. This is a lot earlier than cosine similarity, which only reaches a strongly connected topology with fanout values above 15. Additional results, not plotted for space reasons, also show that the fragmentation induced by the Wup metric is consistently lower than that associated with cosine similarity even for smaller fanout values. With a fanout of 3, for instance, WhatsUp's and CF-Wup's topologies contain respectively an average of 1.6 and 2.6 components, while WhatsUp-Cos's and CF-Cos's contain respectively an average of 12.4 and 14.3.

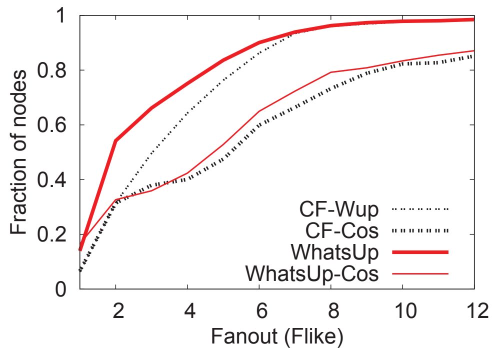


Figure 3.7: *Survey: Size of the LSCC depending on the approach*

3.6.2 Amplification and orientation

Comparing WhatsUp with CF schemes allows us to evaluate the impact of amplification and orientation. The results in Figures 3.6a-3.6f show that WhatsUp consistently outperforms CF, reaching higher F1-Score values with lower fanouts and message costs. Table 3.3 shows that it achieves recall values much higher than those of CF, with less than two thirds the message cost. This is a direct result of the amplification and dislike features, which allow an item to reach interested nodes even after hitting uninterested ones. This observation is confirmed by comparing Figure 3.6e with Figure 3.7. Even if approaches adopting the same metric result in similar topologies as conveyed by Figure 3.7, the performance of those that employ amplification and dislike is consistently higher for corresponding fanout values.

Table 3.4 further illustrates the impact of the *dislike* feature by showing, for each news item received by a node that likes it, the number of times it was forwarded by nodes that did not like it. For instance, we can see that 31% of the news items liked by nodes were forwarded exactly once by nodes that did not like them. This conveys the benefit of the dislike feature and the importance of (negative) feedback from users in giving items a chance to reach interested nodes across the entire network.

Number of dislikes	0	1	2	3	4
Fraction of news	54%	31%	10%	3%	2%

Table 3.4: News received and liked via dislike

Figure 3.8 shows the impact of the TTL value on the performances. Too low a TTL mostly impacts recall; yet values of TTL over 4 do not improve the quality of dissemination. Finally, Table 3.3 also includes the performance of a standard homogeneous gossip protocol, which achieves the worst F1-Score value of 0.51 with almost twice as many messages as WhatsUp.

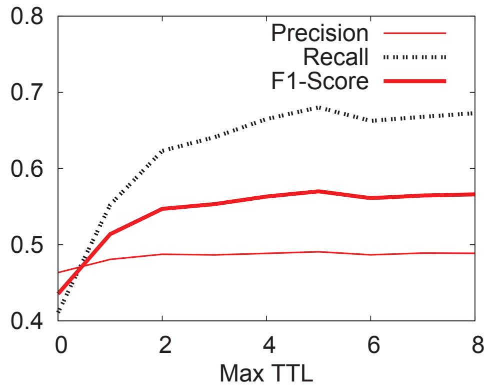


Figure 3.8: *Survey: Impact of the dislike feature of* Beep

Figure 3.9 shows hows nodes at increasing distances from the source of a news item contribute to dissemination. We observe from the bell-shaped curve that most dissemination actions are carried out within a few hops of the source, with an average around 5. This is highly beneficial because a small number of hops leads to news items being disseminated faster.[2] Finally, the plot also confirms the effectiveness of the dislike mechanism with a non-negligible number of infections being due to dislike operations.

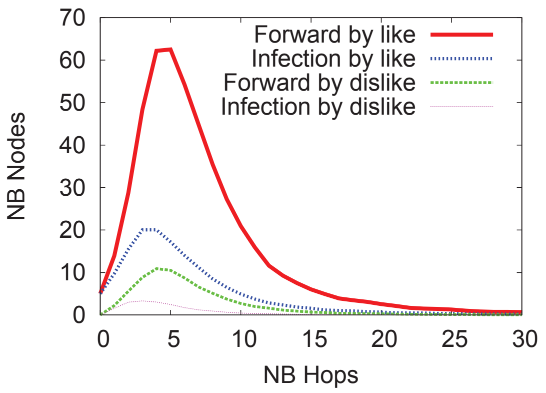


Figure 3.9: *Survey* ($f_{\text{LIKE}} = 5$)*: Impact of amplification of* Beep

3.6.3 Implicit nature of WhatsUp

Next, we evaluate WhatsUp's reliance on implicit acquaintances by comparing it with two forms of *explicit* filtering: *cascading* over explicit social links, and the ideal pub/sub system, C-Pub/Sub.

[2]A precise analysis of dissemination latency would require knowledge of the response time of users. Such an analysis is subject of ongoing work.

The first set of results in Table 3.5 shows that WhatsUp achieves a higher F1-Score with respect to cascading. More specifically, while both approaches provide almost the same level of precision, WhatsUp outperforms (by more than six times) cascading in terms of recall. The very low recall of cascading highlights the fact that the explicit social network does not necessarily connect all the nodes interested in a given topic. The low number of messages of cascading is a result of its small recall. The network traffic per infected user generated by WhatsUp is, in fact, 50% less than that of cascading (2.57K messages vs 5.27K).

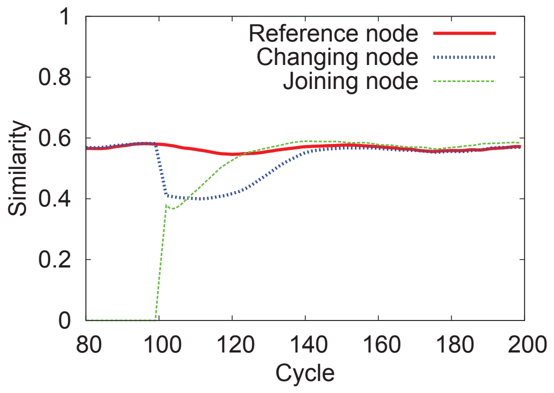
Dataset	Approach	Precision	Recall	F1-Score	Messages
Digg	Cascade	0.57	0.09	0.16	228k
	WhatsUp	**0.56**	**0.57**	**0.57**	**705k**
Survey	C-Pub/Sub	0.40	1.0	0.58	470k
	WhatsUp	**0.47**	**0.83**	**0.60**	**1.1M**

Table 3.5: WhatsUp *vs C-Pub/Sub and Cascading*

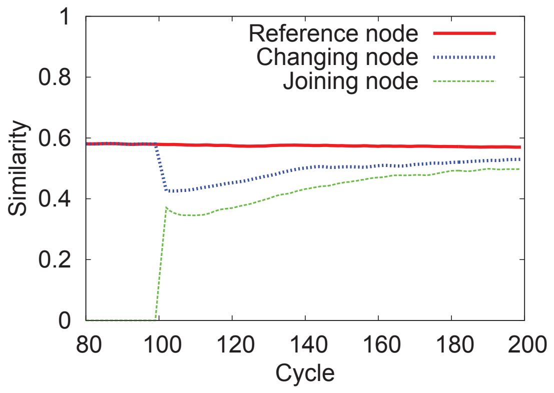
The second set of results in the table compares WhatsUp with C-Pub/Sub. As discussed in Section 3.5.2, C-Pub/Sub disseminates news items to all subscribers with a minimal number of messages. Its recall is therefore 1 while its precision is only limited by the granularity of its topics. In spite of this, WhatsUp improves C-Pub/Sub's accuracy by 12% in the survey dataset with a little more than three times as many messages while conserving a good recall. This results in a better trade-off between accuracy and completeness as indicated by its higher F1-Score.

Another important advantage of Wup's implicit approach is its ability to cope with interest dynamics. To measure this, we evaluate the time required by a new node joining the network and a node changing of interests to converge to a view matching its interests both in WhatsUp (Figure 3.10a) and in WhatsUp-Cos (Figure 3.10b).

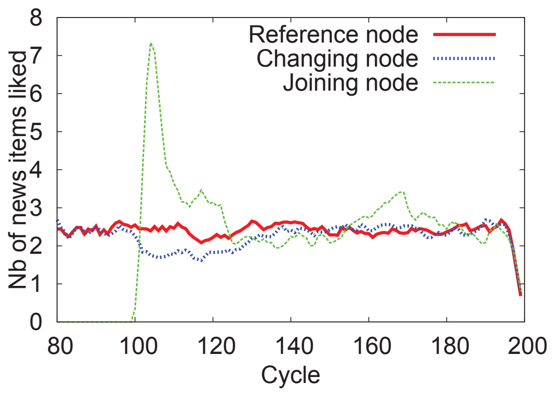
For the joining node, we select a reference node and introduce a new joining node with an identical set of interests. We then compute the average similarity between the reference node and the members of its Wup view and compare it to the same measure applied to the joining node. We repeated the experiment by randomly choosing 100 joining nodes and averaged the results. The Wup metric significantly reduces the number of cycles required by the joining node to rebuild a Wup view that is as good as that of the reference node (20 cycles for WhatsUp vs over 100 for WhatsUp-Cos). Yet, the node starts receiving interesting news quickly as shown in Figure 3.10c. The plot shows a peak in the number of interesting news received as soon as the node joins. This is a result of both our cold start mechanism (Section 3.2.4) and our metric's ability to favor nodes with small profiles. Once the node's profile gets larger, the number of received news per cycle stabilizes to values comparable to those of the reference node. Nonetheless, the joining node reaches 80% of the reference node's precision after only a few cycles.



(a) Similarity in Wup view (WhatsUp)



(b) Similarity in Wup view (WhatsUp-Cos)



(c) Reception of liked news items (WhatsUp)

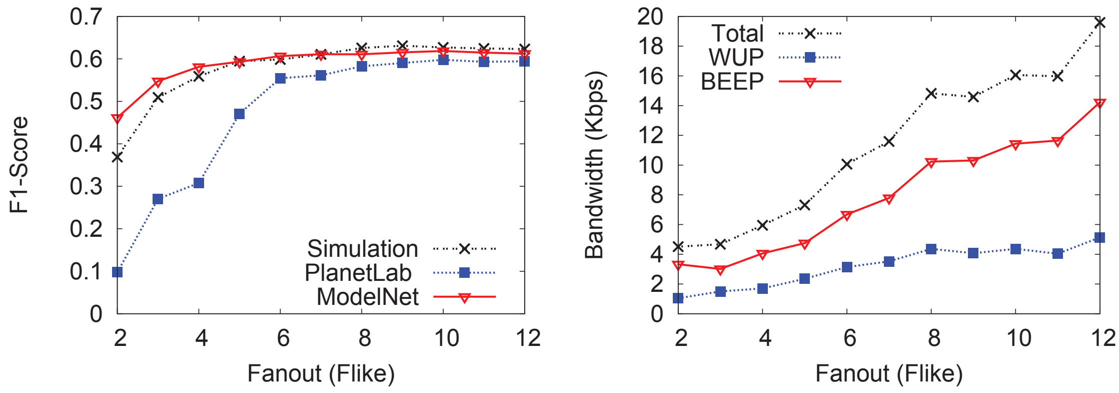
Figure 3.10: *Cold start and dynamics in* WhatsUp

For the changing node, we select a pair of random nodes from the survey dataset and, at 100 cycles into the simulation, we switch their interests and start measuring the time it takes them to rebuild their Wup views. Figure 3.10 displays results obtained by averaging 100 experiments. Again, the Wup metric causes the views to converge faster than cosine similarity: 40 cycles as opposed to over 100. Moreover, the values of recall and precision for the nodes involved in the change of interests never decrease below 80% of the reference node's values. These results are clearly tied to the length of the profile window, set to about 40 cycles in these experiments. Shorter windows would in fact lead to an even more responsive behavior. We are currently evaluating this aspect on the current WhatsUp prototype. Moreover, while it may seem surprising that switching interests takes longer than joining a network from scratch, this experiment is an unlikely situation that provides an upper bound on the impact of more gradual interest changes.

Finally, the implicit nature of WhatsUp and the push nature of Beep also make WhatsUp resilient to basic forms of content bombing. Unless a spammer node has enough resources to contact directly a large number of nodes, it will be unable to flood the network with fake news. The dislike mechanism, with its small fanout and TTL values will, in fact, limit the dissemination of clearly identified spam to a small subset of the network.

3.6.4 Simulation and implementation

We also evaluate the performance obtained by our implementation in two settings: *(i)* a 170 PlanetLab node testbed with 245 users, and *(ii)* an emulated network of 245 nodes (machines and users) deployed on a 25-node cluster equipped with the ModelNet network emulator. For practical reasons we consider a shorter trace and very fast gossip and news-generation cycles of 30sec, with 5 news items per cycle. These gossip frequencies are higher than those we use in our prototype, but they were chosen to be able to run a large number of experiments in reasonable time. We also use a profile window of 4min, compatible with the duration of our experiments (1 to 2 hours each).



(a) Survey: F1-Score (b) Bandwidth in planetlab

Figure 3.11: *Implementation: bandwidth and performance*

Figure 3.11a shows the corresponding results obtained on the survey and compares them to those obtained through simulation on the same 245-user dataset with increasing fanout values. ModelNet results confirm the accuracy of our simulations. The corresponding curves closely match each other except from some fluctuations with small fanout values. PlanetLab results, on the other hand, exhibit a clear decrease in performance with small fanouts. To understand this behavior, we can observe that in simulation and ModelNet, recall reaches scores above 0.50 with fanout values as small as 3. In PlanetLab, it only achieves a value of 0.18 with a fanout of 3, and goes above 0.50 only with fanouts of at least 6. The difference in recall with small fanout values can be easily explained if we observe the message-loss rates in the PlanetLab setting. With a fanout of 3, we recorded that nodes do not receive up to 30% of the news that are correctly sent to them. This is due to network-level losses and to the high load of some PlanetLab nodes, which causes congestion of incoming message queues. The impact of these losses becomes smaller when the fanout increases because Beep is able to produce enough redundancy to recover from the missing messages.

3.6.5 Message loss

To understand the impact of lost messages, we experiment in the ModelNet network emulator with increasing loss rates affecting both Beep and Wup messages and ranging from 0 to a huge value of 50%. Table 3.6 shows that both protocols preserve the reliability properties of gossip-based dissemination. With a fanout of 6, the performance in terms of F1-Score is virtually unchanged with up to 20% of message loss, while it drops only from 0.60 to 0.45 when half of the messages are lost by the network layer. With a fanout of 3, the impact of message loss is clearly more important due to the smaller amount of redundancy. 20% of message loss is sufficient to cause the F1-Score to drop from 0.54 to 0.47. This explains the differences between PlanetLab and ModelNet in Figure 3.11a. These drops are almost uniquely determined by the corresponding recall. With a fanout of 3 and a loss rate of 50%, recall drops to 0.07, causing an artificial increase in precision, and yielding an F1-Score of 0.12, against the 0.45 with a fanout of 6.

Loss Rate	0%		5%		20%		50%	
Fanout	3	6	3	6	3	6	3	6
Recall	0.63	0.82	0.61	0.82	0.46	0.80	0.07	0.45
Precision	0.47	0.48	0.47	0.47	0.47	0.46	0.55	0.44

Table 3.6: *Survey: Performance versus message-loss rate*

3.6.6 Bandwidth consumption

Increasing fanout has a cost, which is highlighted by our bandwidth analysis in Figure 3.11. The number of times each news item is forwarded increases linearly with fanout values, causing an equally linear increase in the bandwidth consumption of Beep. The bandwidth used by Wup also shows a slight increase with fanout due to the corresponding increase in the sizes of the Wup social networks. Nonetheless, the cost of the protocol is dominated by news. This highlights the efficiency of our implicit social-network maintenance. These experiments on a very fast trace with a gossip cycle every 30 sec lead to a bandwidth consumption of about 4 Kbps for Wup's view management. Our prototype is characterized by significantly lower gossip frequencies, on the order of 5 min per gossip cycle. This results in a much lower average bandwidth consumption of about 0.4 Kbps.

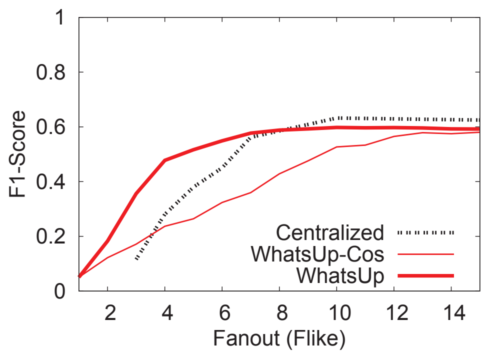


Figure 3.12: *Survey: centralized vs decentralized*

3.6.7 Partial information

To understand the impact of decentralization, we compare WhatsUp with a centralized variant, C-WhatsUp, that exploits global knowledge to instantaneously update node and item profiles. Figure 3.12 shows that WhatsUp provides a very good approximation of this variant (a 5% decrease of the F1-Score). More precisely, global knowledge yields better precision (17%) but slightly lower recall (14%).

Algorithm	Parameters	Precision	Recall	F1-Score
C-WhatsUp	$f_{\text{LIKE}} = 10, f_{\text{DISLIKE}} = 2$	0.53	0.82	**0.64**
User-based CF	$n_{\text{START}} = 5$	0.50	0.47	0.48
Item-based CF	$n_{\text{START}} = 5$	0.53	0.54	0.54
MinHash	$n_{\text{START}} = 3 - 20$	0.75	0.42	0.54
WhatsUp	$f_{\text{LIKE}} = 10$	**0.45**	**0.87**	**0.60**

Table 3.7: WhatsUp *compared with centralized solutions.*

In addition, we also compared WhatsUp against three state-of-the-art recommender systems: User-based CF, Item-based CF, and CF using MinHash clustering. Results are depicted in Table 3.7. The first two improve precision respectively by 11% and 17% over WhatsUp, but decrease the recall respectively by 50% and 40%, ultimately yielding a poorer F1-Score. CF using MinHash clustering [52], on the other hand, uses a cluster-based model to provide better scalability. This improves WhatsUp's precision by 66%, but with a 63% drop in recall. Finally, it is worth mentioning that not only does Wup achieve better performance, but it is also associated with a particularly scalable architecture. Traditional CF systems compute recommendations for all the users in a centralized fashion, which forces them to rely on cloud-based solutions in the majority of cases. The design of WhatsUp instead naturally spreads the computation cost over its users' machines. For each user, the complexity of

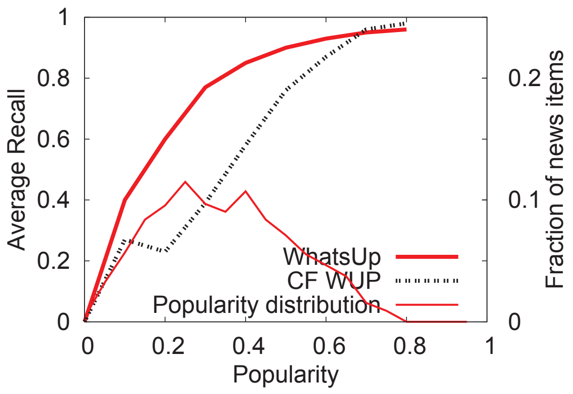


Figure 3.13: *Survey: recall vs popularity*

recommendation depends on the size of its implicit social network and not on the number of users in the entire system.

3.6.8 Sociability and popularity

An additional interesting aspect is the impact of the popularity of items and the sociability of users. Figure 3.13 depicts the distribution of news-item popularity in the survey dataset together with the corresponding recall for WhatsUp and CF-Wup. WhatsUp performs better across most of the spectrum. Nonetheless, its improvement is particularly marked for unpopular items (0 to 0.5). This is highly desirable as popular content is typically much easier to manage than niche content. Recall values appear to converge for very popular items. However, each point in the plot represents an average over several items. An analysis of the data distribution (not shown for space reasons), instead, highlights how CF-Wup exhibits much higher variance leaving some items almost completely out of the dissemination. WhatsUp provides instead good recall values across all items thanks to the effectiveness of its *dislike* feature.

Figure 3.14 instead examines how the F1-Score varies according to the sociability of users in the survey dataset. We define sociability as the ability of a node to exhibit a profile that is close to others, and compute it as the node's average similarity with respect the 15 nodes that are most similar to it. Results confirm the expectations. WhatsUp leverages the similarity of interests between users and provides relevant results for users with alter-egos in the system. The more sociable a node the more it is exposed only to relevant content (improving both recall and precision). This acts as an incentive: the more a user exhibits a consistent behavior, the more she will benefit from the system.

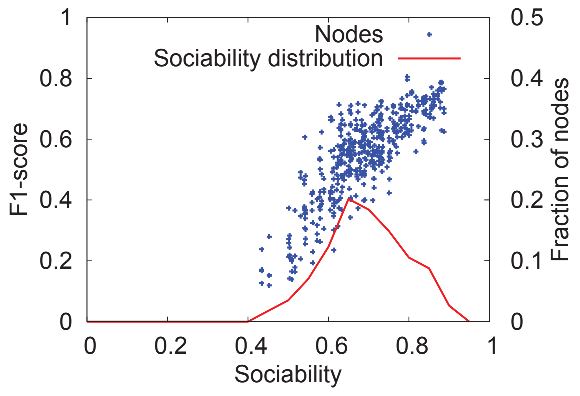


Figure 3.14: *Survey: F1-Score vs sociability*

3.7 Related work

Epidemic protocols [60] are well known to be simple, efficient, and robust means to disseminate information in large-scale systems. So far, they have been mainly homogeneous with respect to fanout and target selection. While some consider an adaptive fanout to control the infection patterns in the network [69, 141], their goal is for messages to reach all nodes, unlike in WhatsUp.

Different approach have been proposed to build recommendation systems. WhatsUp adopts a purely user-based CF strategy for it is more natural to distribute than alternative ones (such as matrix factorization). Determining the most similar users to every user is computationally expensive and usually impossible in real-time for the information stream is huge and changes quickly. Instead, it is typical to cluster users rather than fully leverage the user-centric personalization potential [52]. In this sense, WhatsUp can be seen as a CF scheme producing user-centric recommendations at a small cost through local (P2P) computation and information exchange.

Several metrics have been used to compute the similarity between user profiles. An evaluation of the performance of several metrics on the Orkut social network concluded that cosine similarity shows the best empirical results [132]. In the context of news dissemination, we showed that WhatsUp's metric outperforms cosine similarity. However, different aspects of the recommendation process are difficult to compare in any metric such as recommendation serendipity and the importance of user needs and expectations in a recommender [107]. The trade-off between precision and recall is important. In [153], a topic-diversification approach highlights the importance of serendipity and shows that user satisfaction does not always correlate with high recommender accuracy. WhatsUp's orientation mechanism addresses this issue by balancing precision and recall.

Most of decentralized recommender systems are applied to much less dynamic contexts than instant news. While [77] proposes a Chord-based CF system to decentralize the recommendation database on a P2P infrastructure, it is unclear if it can cope with frequent profile changes and huge continuous streams of items. On the data-sharing front, the fear of the Big-Brother syndrome has also led to decentralized initiatives [83]. However, none of them exploits an implicit social network.

3.8 Concluding remarks

This chapter conveys the feasibility of a fully decentralized collaborative filtering instant news system providing an implicit publish-subscribe abstraction. We did devise and implement such a system: WhatsUp. Our exhaustive experiments show that WhatsUp, while relying only on partial knowledge, inherent to a distributed architecture, achieves a good trade-off between the accuracy and completeness of dissemination.

We had to make several design choices to preserve the simplicity of the system and enable its easy deployment, leaving aside complex or heavyweight alternatives. Yet, leveraging the keywords within news items or ranking them according to users' interest profiles may help refining the filtering. Another observation from our results is the very fact that WhatsUp performs best when user communities are disjoint. While real datasets do not exhibit such communities, an interesting avenue of research would be to investigate solutions that somehow separate communities, potentially allowing nodes to be part of several ones in the form of virtual instances. This is particularly challenging when no explicit classification is available or desirable.

While privacy concerns were out of the scope of this chapter, they might be an issue for users who do not want to disclose their profiles to other users. Integrating a mechanism to protect user profiles from curious users while conserving efficient online personalized dissemination is arduous. In the next chapter, we present obfuscation mechanisms to hide the exact tastes of users and provide a trade-off between the accuracy of recommendation and the disclosure of personal data.

Chapter 4

Privacy-Preserving Distributed Collaborative Filtering

This chapter proposes a privacy preserving mechanism for items dissemination using a decentralized collaborative filtering system. In such a system, a topology reflecting users interests is built and used afterwards by the dissemination algorithm to propagate content while limiting the spam and maximizing the coverage. Users need to exchange their profiles with others in order to compute the topology reflecting their interest. Privacy should be preserved during the topology construction and the dissemination.

We present in this chapter a mechanism to preserve privacy while leveraging user profiles in distributed recommender systems. Our approach relies on (1) an original obfuscation mechanism hiding the exact profiles of users without significantly decreasing their utility, as well as (2) a randomized dissemination algorithm ensuring differential privacy during the dissemination process. We evaluate our system against an alternative providing differential privacy both during profile construction and dissemination. Results show that our solution preserves accuracy without the need for users to reveal their preferences. Our approach is also flexible and more robust to censorship.

4.1 Introduction

Personalization greatly enhances the user experience but induces a trade-off between ensuring user privacy and enabling accurate recommendations [104]. The previous chapter conveys the feasibility of a fully decentralized collaborative filtering. Decentralization removes the monopoly of a central entity that could commercially exploit user profiles. However, users in a decentralized setting may in this case directly access the profiles of other users [137]. Preventing such local privacy breaches is the challenge we address in this chapter. We do so in the context of a generic news item decentralized CF system.

We propose a twofold mechanism: (*1*) an obfuscation technique applied to user profiles, and (*2*) a randomized dissemination protocol satisfying a strong notion of privacy. Each applies to one of the core component of a decentralized user-based CF system: the user-clustering and the dissemination protocols. User clustering builds an interest-based topology (called implicit social network in the previous chapter), implicitly connecting users with similar preferences: it computes the similarity between profiles that capture the opinions of users on the items they have been exposed to. The dissemination protocol propagates the items along the resulting topology.

Our obfuscation technique prevents user machines from exchanging their exact profiles while constructing the interest-based topology. The protocol computes similarities using coarse-grained obfuscated versions of user profiles that reveal only the least sensitive information. To achieve this, it associates each disseminated item with an *item profile*. This profile aggregates information from the profiles of the users that like an item along its dissemination path. This reflects the interests of the portion of the network the item has traversed, gathering the tastes of a community of users that have liked similar items. Our protocol uses this information to construct filters that identify the least sensitive parts of user profiles: those that are most popular among users with similar interests. By controlling the size of these filters, system designers can therefore tune the amount of sensitive information exchanged between users.

Albeit simple and lightweight, this *obfuscation* technique prevents any user from knowing with certainty the exact profile of another user. Moreover, it achieves this without significantly hampering the quality of dissemination: the obfuscated profile reveals enough information to connect users with similar interests.

Our dissemination protocol ensures differential privacy. Originally introduced in the context of statistical databases [61], differential privacy bounds the probability of the output of an algorithm to be sensitive to the presence of information about a given entity – the interests of a user in our context – in the input data. We apply differential privacy by introducing randomness in the dissemination of items. This prevents malicious users from guessing the interests of a user from the items it forwards.

Contributions

To summarize, this chapter presents a simple and efficient mechanism to control the trade-off between privacy and recommendation quality in decentralized collaborative filtering. Our approach relies on (*1*) an original obfuscation mechanism hiding the exact profiles of users without significantly decreasing their utility, as well as (*2*) a randomized dissemination algorithm ensuring differential privacy during the dissemination process.

The rest of this chapter is organized as follows, Section 4.3 presents our system model, Section 4.4 highlights the potential privacy breaches while Section 4.5 and Section 4.6 describe, respectively, our obfuscation mechanism and our randomized dissemination protocol. Section 4.7 presents our experimental set up and Section 4.8 presents our performance evaluation. Section 4.9 concludes this chapter.

4.2 Related work

Privacy is important in many applications. Several approaches [22, 118, 119] apply randomized masking distortion techniques to user profiles to preserve the privacy of sensitive data. However, [81] shows that the predictable structure in the spectral domain of the random distortion can seriously compromise privacy. In the same vein, [90] shows that the variances of the random noises have an important impact on the possibility to filter noise from the original data. In our solution, instead of adding perturbation to user profiles, we exchange with other users a coarse-grain version of this profile only revealing its least sensitive information. The perturbation applied on the profile is not random and depends on the interest of users. As a consequence, separating privacy sensitive information from the distortion becomes more difficult.

[17] designed a statistical measure of privacy based on differential entropy. While this measure can be used to evaluate privacy in a collaborative system [119], it is however difficult to identify the meaning of its value and its implication on the sensitive data. Differential privacy was considered in [61, 76]. In a distributed settings, [24] proposed a differentially private protocol to measure the similarity between peers. While this solution works well in case of static profiles, its differential privacy is not preserved when profiles are dynamic, like in recommendation systems. In addition, still in the context of recommendation systems, [104] highlights the trade-off between privacy and accuracy.

Other decentralized approaches [44, 45, 109] exploit homomorphic encryption approaches in a P2P environment. Homomorphic encryption is a form of encryption where a specific algebraic operation performed on the plain text is equivalent to another algebraic operation performed on the cipher text. In these distributed approaches, this concept is used to measure the similarity between users through an encrypted version of their profiles.

Similarly, [23] proposes an architecture using similar homomorphic cryptosystem to distribute trust on a coalition of servers instead of trusting a single server to provide the recommendation service. Alambic [26], addresses recommendation for e-commerce websites and uses an encryption scheme to split customer data between the merchant and a semi-trusted third party, so that neither can derive sensitive information from their share alone.

The fear of the Big-Brother syndrome has also led to decentralized privacy-preserving initiatives based on anonymity. OneSwarm [83] is an anonymization system for P2P data sharing system using DHT. OneSwarm relies on explicit social networks and allows users to control how their data are exchanged through friends. Data is transferred through a mesh of untrusted and trusted peers populated from user social networks. Gossple [32], a fully decentralized anonymous collaborative network, also addresses privacy by anonymity. Authors design a gossip on behalf protocol is used to hide the association between a user and its profiles.

However, the privacy guaranty provided by anonymity can be limited in a static setting if an adversary is able to access both anonymized and plain information. For instance, although the Netflix dataset was anonymized, users have been identified by matching their anonymized ratings with the ratings of IMDB [112].

Lastly, in [82], privacy is addressed by trust. Participants exchange information only across content producer/consumer pairs. Producers share their profiles only with the consumers to which they deliver content, and accordingly consumers share their profile only with the producers to which they rate their contents.

4.3 System Model

We consider here a decentralized user-based collaborative filtering (CF) news item recommender a la WhatsUp, using the same underlying techniques. However, in order to observe its behaviour in a classical setting, we apply it in a generic case, means without the amplification and orientation mechanisms of Beep. We remind below the two core components in this CF: *user clustering*, and *item dissemination.*

User clustering aims at identifying the k-nearest neighbors of each user. To achieve this, it associates each user with a profile who exchanges with others and then computes similarities between the profiles of pairs of users. A decentralized gossip-based approach is used to build and to maintains an interest-based overlay topology where each user is connected to users with similar interests.

Dissemination also operates in a fully decentralized manner. Here, we consider a simple epidemic protocol: a user that generates an item, or that expresses a positive opinion on a received one, forwards it to her k interest-based neighbors.

4.4 Privacy Breaches in Distributed CF

The presented CF system does not integrate specific mechanisms to protect the privacy of users. While decentralization removes the prying eyes of *Big-Brother* companies, it leaves those of curious users who might want to discover the personal tastes of others. In this chapter, we aim to protect a decentralized item recommender from malicious nodes that extract information in the two following ways: (*i*) from the profiles they exchange with other nodes; and (*ii*) from the items they receive in the dissemination process.

As described previously, nodes exchange profile information for maintaining the interest-based overlay. Profiles contains precise information about the interests of a user, which are potentially sensitive. Similarly, the dissemination protocol is driven by the interest of users. As a consequence, a node a that receives an item from another node n can conclude that n *liked* that item. The predictive nature of this dissemination protocol thus also constitutes a leak of sensitive information.

The remainder of this chapter presents a solution that addresses each of these two vulnerabilities by means of two dedicated components. The first is a profile obfuscation mechanism that prevents curious users from ascertaining the tastes of the user associated with an exchanged profile. The second, is a randomized dissemination protocol that hides the preferences of nodes during the item forwarding process.

4.5 Obfuscation Mechanism

Our first contribution is an obfuscation protocol that protects user profiles by (i) aggregating their interests with those of similar users, and (ii) revealing only the least sensitive information to other users. By tuning these two mechanisms, system designers can manage the trade-off between disclosed information and recommendation quality [104]. An excessively obfuscated profile that reveals very little information is difficult to compromise, but it also provides poor recommendation performance. Conversely, a highly accurate profile yields better recommendations, but does not protect privacy-sensitive information effectively. As we show in Section 4.8, our obfuscation mechanism provides good recommendation while protecting privacy.

4.5.1 Overview

Figure 4.1 provides an overview of our protocol. Each node maintains a profile that lists its opinions on the items it has been exposed to within the latest *time window*. We name it *private profile* because a node never shares its content with anyone else. In addition, we associate each item with an *item profile*, and each node, n, with three new profile structures: two private ones, the *compact profile* and the *filter profile*; and the *obfuscated profile* potentially shared. The *item profile* aggregates the interests of users who liked an item. The *compact profile* provides a summary of node n's interests.

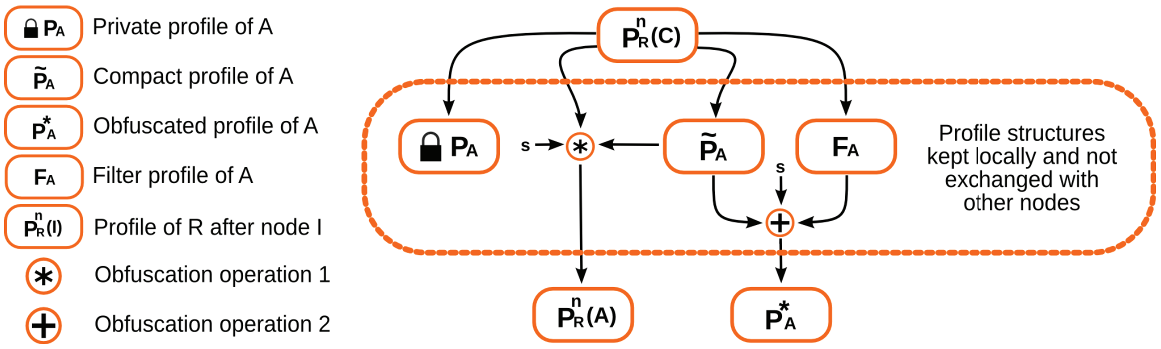


Figure 4.1: *Operations performed on profiles when node A likes a received item R from C.*

The *filter profile* captures the main interest trends among the nodes that are similar to n. Finally, the *obfuscated profile* provides a filtered version of n's compact profile containing only the least senstitive information. Nodes do not strictly maintain the *compact*, *filter*, and *obfuscated profiles*: they are computed when needed, using the information from the *private* and *item* profiles.

4.5.2 Profile Updates

Private Profile A node updates its private profile whenever it expresses a like/dislike opinion about an item (line 37 and 42 in Algorithm 3), or when generating a new item. In either case, the node inserts a new tuple into its private profile. For liked items, this tuple contains the item identifier, its timestamp – indicating when the item was generated, a score value – 1 if the node liked the item, 0 otherwise, the item vector, and the item profile upon receipt. For a disliked item, the tuple contains only the item identifier, while the remaining fields are empty. Only the *private profile* gathers all the information on the interests of a user. Our obfuscation mechanism never reveals its contents to other nodes: it only uses it to update the other data structures as we discuss in the following.

Compact Profile Unlike private profiles, which contain item identifiers and their associated scores, compact profile stores liked items in the form of a d-dimensional bit array. This compact profile is produced using Random Indexing, an incremental dimension reduction technique [89, 145]. The basic idea of Random indexing is to reduce the private profile of users to a compact profile accumulating the random signatures, called *item vector*, of each liked item.

An item vector is generated by the source of an item and it consists of a sparse d-dimensional bit arrays with small number of randomly distributed 1's, with the rest of the elements set to 0. Several methods exist to build these vectors [36, 89]. We consider a vector size of d bits and a number $b << d$ of 1 values as system-wide parameters. The source of an item simply generates b distinct array positions and sets

Algorithm 3: Receiving an item.

```
35 on receive (item $< id^N, t^N >$, item vector $S^N$, item profile $P^N$) do
36     if iLike($id^N$) then
37         $P \leftarrow < id^N, t^N, 1, S^N, P^N >$
38         buildCompactProfile($S^N$)
39         updateItemProfile($P^N$)
40         forward($< id^N, t^N >$, $S^N$, $P^N$)
41     else
42         $P \leftarrow < id^N, t^N, 0 >$
43 function buildCompactProfile()
44     for all $< id, t, 1, S, P^N > \in P$
45         $\tilde{P}[i] = S[i]\ OR\ \tilde{P}[i]$
46 function updateItemProfile(item vector $P^N$)
47     for all $i \in P^N$
48         $Sum[i] = Integer(\tilde{P}[i]) + Integer(P^N[i])$
49     for all $i \in$ the $s$ highest values in $Sum$
50         $P^N[i] = 1$
51 function forward($< id^R, t^R >$, item vector $S^N$, item profile $P^N$)
52     for all $n \in Neighbors$
53         send $< id^R.t^R >$ with associated $S^N$ and $P^N$ to $n$
```

the corresponding bits to 1. It then attaches the resulting vector to the item before disseminating it. Finally, the compact profile of a user, u, is computed as the bitwise OR of the item vectors of the items u liked.

As shown in Figure 4.1 and on lines 48 of Algorithm 3 and 58 of Algorithm 4, a node uses the compact profile both to update the item profile of an item it likes and to compute its obfuscated profile when exchanging clustering information with other nodes. In each of these two cases, the node computes a fresh compact profile as the bitwise OR of the item vectors in its private profile (line 45 of Algorithm 3). This on demand computation allows the compact profile to take into account only the items associated with the current *time window*. It is in fact impossible to remove an item from an existing compact profile. The reason is that compact profile provides a first basic form of obfuscation of the interests of a user through bits collisions: a bit with value 1 in the compact profile of a node may in fact results from any of the liked items whose vectors have the corresponding bit set. By changing the b and d parameters of item vectors, system designers can vary the collision rate between item vectors according to the number of items in the system and thus tune the efficacy of this obfuscation step.

The use of Random Indexing has two clear advantages. First, the presence of bits collisions makes it harder for attackers to identify the items in a given profile. Second, the fixed and small size of bit vectors limits the size of the messages exchanged by the nodes in the system. As evaluated, in Section 4.8.7, this drastically reduces the bandwidth cost of our protocol.

Algorithm 4: Building obfuscated profile.

```
54 on demend do
55     Algorithm1.buildCompactProfile()
56     buildFilterProfile()
57     for all $i \in$ the $s$ highest values in $F$
58         $P^*[i] = \tilde{P}[i]$

59 function buildFilterProfile()
60     for all $< id, t, 1, S, P^N > \in P$
61         $F[i] = F[i] + Integer(P^N[i])$
```

Item Profile A node never reveals its compact profile. Instead, it injects part of it in the item profiles of the items it likes. Consequently, the item profile of an item gathers the interests of the users that liked the item along its dissemination path. In other words, it reflects the interests of the portion of the network the item has traversed. A parameter s controls how much information from the compact profile nodes include in the item profile.

More precisely, let n be a node that liked an item R. When n receives R for the first time, it first computes its compact profile as described above. Then, n builds an integer vector as the bit-by-bit sum of the item profile and its own compact profile (line 48 in Algorithm 3). Each entry in this vector has a value in $\{0, 1, 2\}$: node n chooses the s vector positions with the highest values, breaking ties randomly, and creates a fresh profile for item R by setting the corresponding bits to 1 and the remaining ones to 0. Finally, when n generates the profile for a new item (line 50 in Algorithm 3), it simply sets to 1 the values of s bits from those that are set in its compact profile. This update process ensures that each item profile always contains s bits with value 1.

Filter Profile Nodes compute their filter profiles whenever they need to exchange clustering information with other nodes (line 56 in Algorithm 4). Unlike the other profiles associated with nodes, this profile consists of a vector of integer values and does not represent the interests of a user. Rather it captures the interests of the community of users that have liked similar items. A node computes the value at each position in its filter profile by summing the values of the bits in the corresponding position in the profiles of the items it liked (line 61 in Algorithm 4) in the latest *time window*. This causes the filter profile to record the popularity of each bit within a community of nodes that liked similar items.

Obfuscated Profiles A node computes its obfuscated profile whenever it needs to exchange it with other nodes as part of the clustering protocol. As shown in Figure 4.1, it achieves this by filtering the contents of its compact profile using its filter profile: this yields a bit vector that captures the most popular bits in the node's community and thus hides its most specific and unique tastes. The fine-grained information contained in the node's private and compact profiles remains instead secret throughout the system's operation.

As shown on line 55 and line 56 of Algorithm 4, a node n computes its obfuscated profile by first generating its compact and filter profiles as described above. Then it selects the s positions that have the highest values in the filter profile, breaking ties randomly, and sets the corresponding bits in the obfuscated profile to the values they have in its compact profile. It then sets all the remaining bits in the obfuscated profile to 0.

The resulting profile has s bits (set at 0 or 1) reflecting the node's compact profile providing a coarse-grained of user interests. Through the value of s, the system designer can control the amount of information that can filter from the compact to the obfuscated profile, and can therefore tune the trade-off between privacy and recommendation quality. It is important to note that the position of the bits at 1 in the obfuscated profile are not correlated to the item vectors of item that user liked but depends on the filter profile. Indeed, only one bit of an item vector can be selected by the filter profile. This prevents isolated attackers from precisely understanding which news items the node liked as shown in Section 4.8.5.

4.6 Randomized Dissemination

As described in Section 4.4, an attacker can discover the opinions of a user by observing the items she forwards, because of the predictability of the dissemination protocol. We reduce this vulnerability by introducing some randomness in the dissemination process: a node that likes an item will drop it with probability pf, while a node that does not like it will forward it with the same pf. Thanks to this randomization, an attacker cannot deduce with certainty whether a user liked an item from the observation of the items she forwards.

It is important to note that this randomization only affects dissemination. Thus, nodes continue to update their profiles and those of the items as specified in Section 4.5. In particular, a node will not update the profile of an item it disliked, even if it decides to forward it. Randomization reduces the information leaked during dissemination at the cost of a decrease in accuracy. Indeed, by forwarding disliked items, and dropping liked items, nodes reduce the system's ability to filter and promote relevant content. We evaluate this trade-off in Section 4.8.5. In the following, we show that this randomization component is differentially private [61].

A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if it produces approximately the same output when applied to two neighboring datasets (*i.e.* which differ on a single element). In the context of dissemination, the datasets that need to be randomized are vectors of user opinions. Given two neighboring vectors of opinions (*i.e.* differing on a single opinion) $o1 \in \mathcal{D}^n$ and $o2 \in \mathcal{D}^n$, we define differential privacy as follows.

Definition 1 (Differential privacy [62])**.** *A randomized function $\mathcal{F} : \mathcal{D}^n \to \mathcal{D}^n$ is ϵ-differentially private, if for any pair of neighboring opinion vectors $\mathbf{o1}, \mathbf{o2} \in \mathcal{D}^n$ and* *for all* $\mathbf{t} \in \mathcal{D}^n$*:*

$$\Pr[\mathcal{F}(\mathbf{o1}) = \mathbf{t}] \leq e^{\epsilon} \cdot \Pr[\mathcal{F}(\mathbf{o2}) = \mathbf{t}]$$

This probability is taken over the randomness of $\mathcal{F}$*, while* e *is the base of the natural logarithm.*

In the case of our algorithm, we toss a coin each time the user expresses her opinion about an item in order to decide whether the item should be forwarded. This scheme is known as randomized response [146]: instead of randomizing the output of a function f, we randomize each of its inputs independently. Because these inputs as well as the output values are binary $\in \{0, 1\}$, we can rewrite the above equation as follows.

$$\Pr[f(o) = b] \leq e^{\epsilon} \cdot \Pr[f(1 - o) = b]$$

Our randomization function f flips the opinion o and produces the output $1 - o$ with probability pf. In order to achieve ϵ-differential privacy the value of pf must be such that:

$$1/(e^{\epsilon} + 1) \leq pf \leq 1/2$$

For space reasons, we omit the details of the reasoning leading to this result, as well as the proof of the equivalence between randomized response and Definition 1. Nonetheless they are similar to those in [24].

This algorithm reduces the amount of information an observer gets when receiving an item from a user. Instead of knowing with certainty that the user liked the item, the observer knows that the user liked it with probability $1 - pf$. However, this does not make our solution differentially private. The dissemination component is, but it only ensures ϵ-differential privacy when a user expresses her opinion about an item, not when she generates a new one.

4.7 Experimental setup

We implemented and extensively evaluated our approach using a real dataset from a user survey. We also compare our solution with a baseline solution with no privacy mechanism, where profiles are exchanged in clear, and a solution that applies a differentially private mechanism both when generating the profiles that users exchange and upon dissemination. We refer to our solution as OPRD (Obfuscation Profile and Randomized Dissemination) in the following.

4.7.1 Dataset

To evaluate our approach against a real dataset, we conducted a survey on 200 news items involving 120 colleagues and relatives. We selected news items randomly from a set of RSS feeds illustrating various topics (culture, politics, people, sports,...). We exposed this list to our test users and gathered their opinions (like/dislike) on each news item. This provided us with a small but *real* dataset of users exposed to exactly the same news items. To scale out our system, we generated 4 instances of each user and news item in the experiments. While this may introduce a bias, this affects accuracy of both our mechanisms and the solutions we compare against.

4.7.2 Alternatives

We compare our approach with the two following alternatives.

Cleartext profile (CT) This baseline approach implements the decentralized CF solution presented in Section 4.3 where user profiles are exchanged in clear during the clustering process. This solution does not provide any privacy mechanism.

Differentially private approach (2-DP) This alternative, denoted by 2-DP in the following, applies randomization both when generating user profiles and during dissemination. Every time a user expresses an opinion about an item, the algorithm inverses it with probability *pd*: this results in a differentially private clustering protocol and a differentially private dissemination protocol. The latter is similar to our randomized dissemination. However, unlike our solution, 2-DP also applies randomness when generating user profiles. When a user dislikes an item, 2-DP considers this item as liked with a probability *pd*, thus integrating it in the profile of the user and disseminating it to her neighbors. Conversely, when a user likes an item, 2-DP considers it as disliked with probability *pd*. In this case, it silently drops it without including it in the user's profile.

2-DP builds user profiles that are structurally similar to our compact profiles. However, they gather the item vectors of the items identified as liked after the randomization of user opinions. This extends the privacy guarantee associated with our dissemination protocol to the profiles of users. This represents a contribution in its own right. For space reasons, we do not include the associated proof. However, it follows a similar intuition than the one presented in Section 4.6.

As user profiles change over time and are impacted by the dissemination of items, applying a randomization function on cleartext profiles as in [24] is not enough. Iteratively probing the profiles of a user and analyzing the dissemination process could be enough to weaken the privacy guarantee. Instead, 2-DP does not randomize profiles, but it randomizes the opinion of a user on the items she is exposed to. Moreover, it does so independently of the user's opinion on other items.

2-DP uses the output of its randomization function to build user profiles and drive the dissemination. In particular, users use the resulting randomized profiles to compute their clustering views. We show in Section 4.8.4 that this introduces a weakness in the context of the decentralized CF scheme considered in this work.

4.7.3 Evaluation metrics

Accuracy

We evaluate accuracy along the traditional metrics used in information-retrieval systems presented in Section 2.2.3: *recall*, *precision*, and *F1-Score* . Briefly, the precision reflects the accuracy of the recommendation defined as the number of interesting items received over the total number of received items while the recall expresses its completeness as the number of interested items received over the number of items which should be received. The F1-Score, in turn, captures the trade-off between these two metrics and is defined as the harmonic mean of precision and recall [140].

System

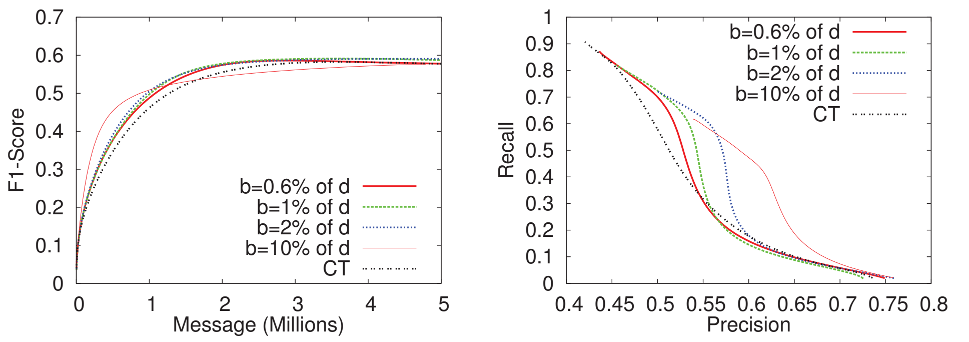
We evaluate the overhead of the system in terms of the network traffic it generates. For simulations, we compute the total number of sent messages. For our implementation, we instead measure the average consumed bandwidth. A key parameter that determines network traffic is the *fanout* of the dissemination protocol, *i.e.* the number of neighbors from the interest-based overlay to which nodes forward each item.

Privacy

We define privacy as the ability of a system to hide the profile of a user from other users. We measure it by means of two metrics. The first evaluates to what extent the obfuscated profile is close to the real one by measuring the similarity between the two. We consider the Jaccard index [140] to measure the similarity between a compact profile and the corresponding obfuscated one. The second measures the fraction of items present in a compact profile out of those that can be predicted by analyzing the presence of item vectors in the corresponding obfuscated profile. As item vectors are public, a malicious user can leverage them to guess the contents of the obfuscated profiles of other users, thereby inferring their interests.

4.8 Performance evaluation

In this section, we evaluate the ability of our solution to achieve efficient information dissemination while protecting the profiles of its users. First, we show that compacting user profiles, filtering sensitive information, and randomizing dissemination do not significantly affect the accuracy of dissemination when compared to CT, yielding slightly better results than 2-DP. Then we analyze the trade-off between accuracy and privacy and show the clear advantage of our solution in protecting user profiles in the context of a censorship attack. Finally, we show the benefits of our solution in term of network cost. We conducted an extensive evaluation through simulations, and



(a) F1-Score vs number of messages for various b-to-d ratios (b) Precision-recall curve for various b-to-d ratios

Figure 4.2: *Impact of compacting the profiles*

through a real implementation deployed on PlanelLab. In both cases, we randomly select the source of each item among all users.

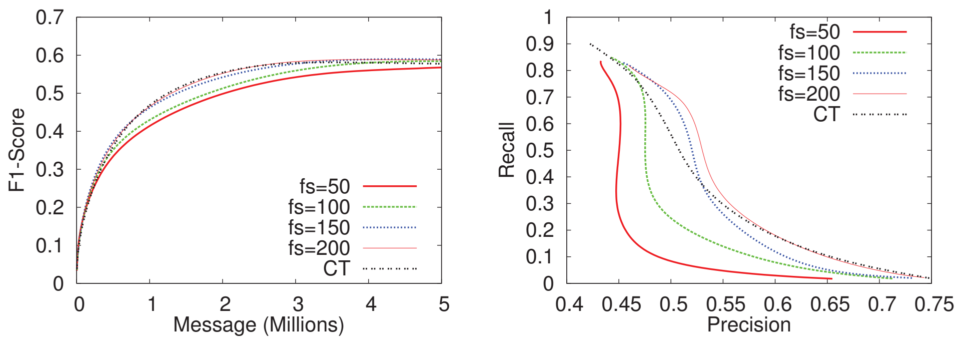
4.8.1 Compacting profiles

As explained in Section 4.5.2, our solution associates each item with a (sparse) item vector containing b ones out of d possible positions. When a user likes an item, we add the corresponding item vector to her compact profile by performing a bitwise OR with the current profile. The ratio between b and d affects the probability of having the same bits at 1 in the vectors of two different items in a given compact profile. Figure 4.2 evaluates its effect on performance.

Figure 4.2a shows the values of the F1-Score with increasing network traffic for various values of the b-to-d ratio. The points in each curve correspond to a range of fanout values, the fanout being the number of neighbors to which a user forwards an item she likes: the larger the fanout the higher the load on the network. Figure 4.2b shows instead the corresponding precision-recall curve. Again, each curve reflects a range of fanout values: the larger the fanout, the higher the recall, and the lower the precision.

Interestingly, the larger the values of the b-to-d ratio, the bigger the difference between our solution and CT. When the b-to-d ratio is low, it is very unlikely for any two item vectors to contain common bits at 1. As a result, the performance of our solution closely mimics that of CT. When the b-to-d ratio increases, the number of collisions between item vectors – cases in which two distinct item vectors have common bits at 1 – also increases. This has two interesting effects on performance.

The first is that the F1-Score increases faster with the fanout and thus with the number of messages: the $b = 10\%$ curve climbs to an F1-Score of 0.4 with less than $400k$ messages. The curve on Figure 4.2b shows that this results from a higher recall for corresponding precision values (bump in the $b = 10\%$ curve). The high probability



(a) F1-Score vs number of messages for various filter sizes (fs)

(b) Precision-recall curve for various size of filters (fs)

Figure 4.3: *Impact of filtering sensitive information*

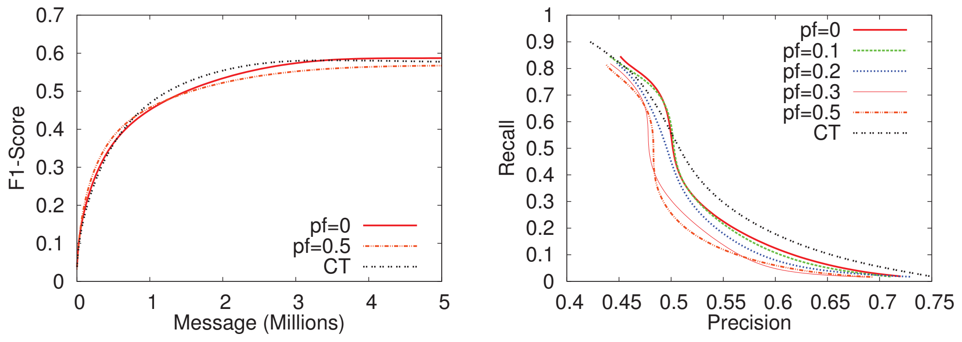
of collisions between item vectors results in some profiles being similar even if they do not contain many common items. The resulting more clustered topology facilitates dissemination and exposes nodes to more items. This makes it easier for them to discover real similar interests and therefore good neighbors even at low fanout values.

However, the second effect is that the maximum F1-Score attained by the protocol with a large b-to-d ratio (to the right of Figure 4.2a) stagnates to lower values. Figure 4.2b clarifies that this results from lower recall values, as indicated by the left endpoints of the curves corresponding to high values of b. The reason for this behavior is that the clustered topology resulting from large values of b, advantageous with small fanout values (small number of messages), ultimately inhibits the dissemination of items to large populations of users. This effect is even more prominent when considering values of b for which compact profiles contain a vast majority of bits set to 1 (not shown in the plot).

In the following, we set d to 500 and b to 5 for our evaluations. System designers should set these parameters according to the rate at which items are generated to achieve the desired rate of collisions in compact profiles.

4.8.2 Filtering sensitive information

In our solution, the size of the filter defines how much information from the compact profile appears in the obfuscated profile. The larger the filter, the more the revealed information. Figure 4.3a depicts the F1-Score as a function of the number of messages. Clearly, performance increases with the size of the filter. The precision-recall curve in Figure 4.3b shows that this results mainly from an increase in precision. The important aspect is that both plots highlight that a filter of 200 bits (*e.g.* 40% of the compact profile) achieves performance values similar to those of a system using full profiles.



(a) F1-Score vs number of messages for various pf ($fs = 150$) (b) Precision-recall curve for various pf ($fs = 150$)

Figure 4.4: *Impact of obfuscating profiles and randomizing dissemination*

4.8.3 Randomizing the dissemination

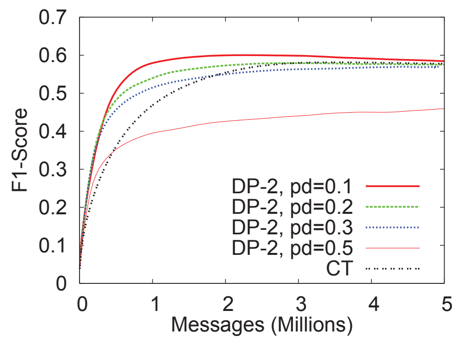
We now evaluate the impact of randomizing the dissemination process in addition to the obfuscation mechanism evaluated above (previous results were obtained without randomization). This removes the predictive nature of dissemination. Figure 4.4a shows the F1-Score for our solution using a filter size of 150 and several values for pf. Performance decreases slightly as we increase the amount of randomness (for clarity, we only show $pf = 0$ and $pf = 0.5$, the other curves being in between). Figure 4.4b shows that increasing pf results mostly in a decrease in precision.

4.8.4 Evaluating 2-DP

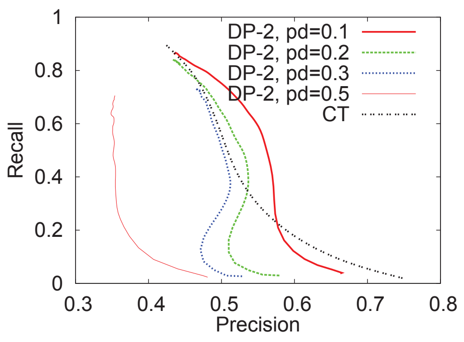
In this section, we evaluate the 2-DP alternative defined in Section 4.7.2. 2-DP reverses the behavior of users with a probability, *pd*, that affects both the construction of user profiles and the dissemination process. This differs from our solution, which only uses a differentially private dissemination protocol.

Figure 4.5a shows the F1-Score of 2-DP versus network traffic for various values of *pd*. While for $dp = 0$, 2-DP is equivalent to CT, performance at low fanout values increases for $dp = 0.1$, but decreases again for larger values of *dp*. A small amount of randomness proves beneficial and allows the protocol to achieve a better compromise between recall and precision at low fanouts. This effect, however, disappears when the number of messages increases at high fanouts. Too much randomness, on the other hand, causes a drastic decrease in the F1-Score. Figure 4.5b shows that randomness induces an increase in recall with respect to CT and a decrease in precision. The former dominates with low values of *pd* while the latter dominates for high values.

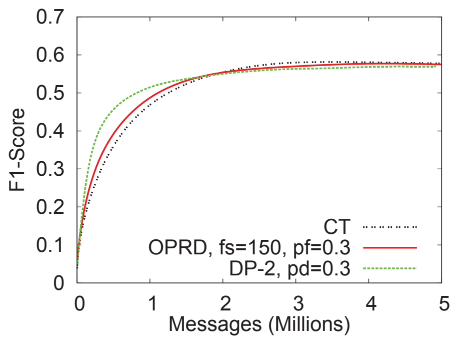
Figure 4.5c compares the F1-Score of OPRD using a filter of size of 150 and a pf value of 0.3, with that of CT and 2-DP using a *pd* of 0.3. We observe that above $2M$ messages, our solution provides slightly better F1-Score values than 2-DP. Overall,



(a) 2-DP: F1-Score vs number of messages for various *pd*



(b) 2-DP: Precision-recall curve for various *pd*



(c) OPRD vs 2-DP: F1-Score vs number of messages

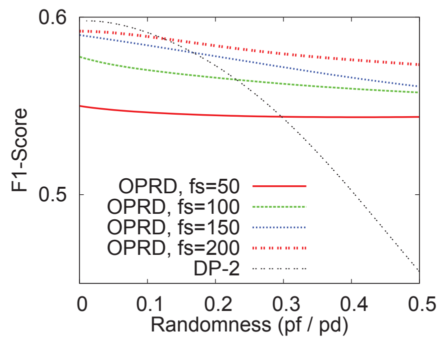
Figure 4.5: *Differential private alternative*

however, the best performances of the two approaches are comparable. In the following, we show that this is not the case for their ability to protect user profiles.

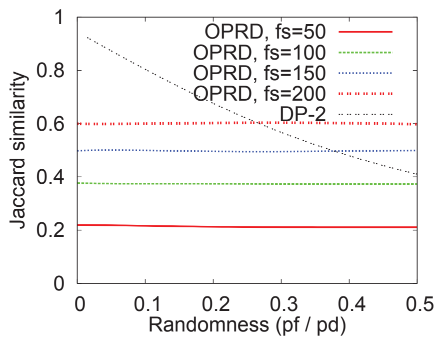
4.8.5 Privacy versus accuracy

We evaluate the trade-off between the privacy, measured as the ability to conceal the exact profiles of users, and the accuracy of both our solution and 2-DP. In our solution, this trade-off is controlled by two parameters: the size of the filter, and the probability pf to disseminate a disliked item. 2-DP controls this trade-off by tuning the probability pd to switch the opinion of the user, impacting both profile generation and the dissemination process.

Figure 4.6a compares the recommendation performance by measuring the F1-Score values for various filter sizes. The x-axis represents the evolution of the probabilities pf, for our solution, and pd, for 2-DP. We show that the F1-Score of 2-DP decreases faster than ours. The F1-Score of 2-DP with a pd of at least 0.2 is smaller than that

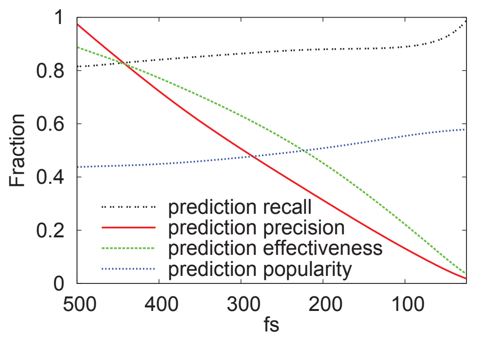


(a) Accuracy: OPRD vs 2-DP

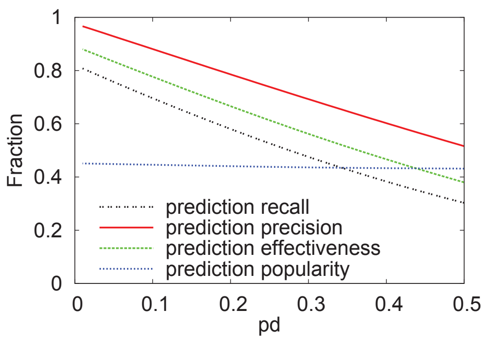


(b) Similarity between public and real profiles

Figure 4.6: *Randomness vs performance and level of privacy*



(a) Prediction with OPRD, $pf = 0.2$



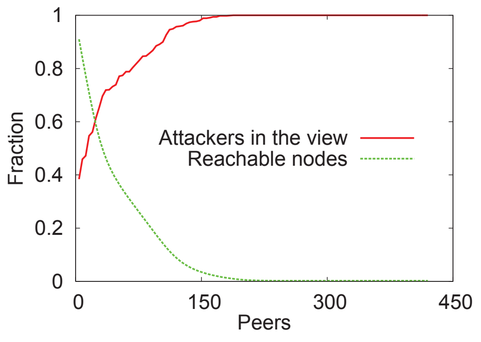
(b) Prediction with 2-DP

Figure 4.7: *Profile prediction*

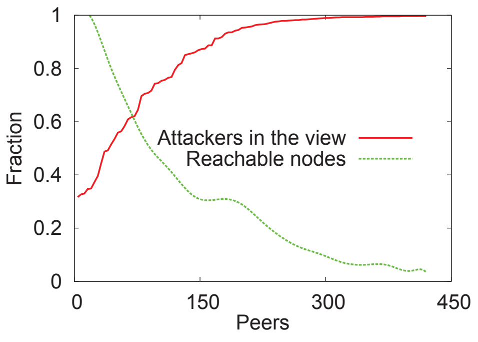
of our solution with a filter size greater than 100. In addition, revealing the least sensitive 10% of the compact profile ($fs = 50$) yields better performance than 2-DP with $pd \geq 0.3$.

Figure 4.6b measures the level of privacy as the similarity between the compact profile and the obfuscated profile using the Jaccard index: lower similarity implies more privacy. Results show the flexibility of our solution. As our randomized dissemination protocol hardly impacts the obfuscated profile, our results are almost independent of pf. 2-DP sees instead its similarity decrease with increasing pd. With $pd = 0.3$, 2-DP yields a similarity value of about 0.55 with an F1-Score (from Figure 4.6a) of 0.56. Our approach, on the other hand yields the same similarity value with a filter size between $150 < fs < 200$, which corresponds to an F1-Score value of about 0.58.

Figure 4.7, instead, assesses privacy by measuring if the items in a user's real profile can be predicted by analyzing the item vectors in profile exchanged with neighbors.



(a) Censorship: 2-DP, $pd = 0.2$



(b) Censorship: OPRD, $fs = 125$, $pf = 0.2$

Figure 4.8: *Resilience to censorship*

Note that in the 2-DP, the real profile is the one that would exist without random perturbations. We evaluate this aspect by measuring the recall and the precision of each prediction. Prediction recall measures the fraction of correctly predicted items out of those in the compact profile. Prediction precision measures the fraction of correct predictions out of all the prediction attempts. For our solution, in Figure 4.7a, we use a $pf = 0.2$ to control the randomized dissemination, and vary the filter size. For 2-DP (Figure 4.7b), we instead vary pd.

The plots show that while predictions on our approach can be fairly precise, they cover only a very small fraction of the compact profile with reasonable values of fs. With $fs = 200$, which gives similar performance as using the entire compact profile (Figure 6.2), the prediction recall is of about one third. In contrast, 2-DP exposes a higher number of items from the compact profile. With $pd = 0.2$ the prediction recall is of 0.8 with a prediction precision of 0.6. The curves for prediction effectiveness, computed as the harmonic mean of recall and precision, further highlight our approach's ability to strike an advantageous balance between privacy and recommendation performance.

The two plots also show the average popularity of the predicted items. It is interesting to observe that with small enough filters, we predict the most popular items, which are arguably the least private. Finally, we also observe that the compact profile itself provides a small protection to the prediction of items due to its inherent collision rate. With a filter of size 500 (*e.g.* with no difference between the compact and the public profile), the error rate is equal to 0.15.

4.8.6 Illustration: Resilience to censorship attack

Exposing user profiles enables potential censorship attacks. In this section, we evaluate the resilience of our obfuscation mechanism against censorship by implementing a simple eclipse attack [130]. In this attack, a coalition of censors mirrors the (obfuscated) profile of a target node in order to populate its clustering view. This is turn isolates it from the remaining nodes since its only neighbors are all censors. If the user profiles are exposed in clear, the profile of the censors matches exactly that of the target node: this gives censors a very high probability to enter its view. Once the censors have fully populated the target node's view, they simply intercept all the messages sent by the target node, preventing their dissemination. To evaluate the efficiency of this attack, we use two different metrics. The first is the poisoning rate of the target's clustering view by attackers. The second one is the fraction of honest nodes (*e.g.* not censors) in the system that the target can reach when it sends an item.

We ran this attack for each user in the dataset. The x-axis represents the users in the experiment sorted by their sensitivity to the attack. Figure 4.8a and Figure 4.8b depict the results obtained with a cluster size of 50 and 50 censors (we observe similar results independently of the cluster size). In addition, this experiment uses a filter of 125 and $pf = 0.2$ for our solution, and $pd = 0.2$ for 2-DP. We can clearly see that 2-DP is not effective in preventing censorship attacks: only 150 nodes have a poisoning rate lower than 1. This is due to the fact that in order to ensure differential privacy, 2-DP, randomizes the profiles once so that they cannot be iteratively probed. However, this forces 2-DP to use the same profile internally and during the exchanges for similarity computation. Therefore 2-DP exhibits the exact same vulnerability as CT. The profiles of the censors can trivially match the target node.

Our approach is much more resilient to this specific censorship attack. For instance, it is almost impossible for censors to intercept all the messages sent by the target and only a third of the nodes have a fully poisoned clustering view. The obfuscated profile only reveals the least sensitive information to other nodes: censors only mirror a coarse-grained sub part of the target's profile. As a consequence, they are more likely to have a profile similar to users with similar interests than to match the one of the target. This observation is confirmed by Figure 4.6b which shows the difference in terms of similarity between the obfuscated profile and the compact profile of users. The resilience of OPRD to this censorship attack is driven by the size of the obfuscation filter, the smaller the filter, the more resilient the protocol.

4.8.7 Bandwidth consumption

We also conducted experiments using our prototype with 215 users running on approximately 110 PlanetLab nodes in order to evaluate the reduction on the network cost due to the dimension reduction of our profiles. The results in terms of F1-Score, recall, and precision closely mimic those obtained with our simulations and are therefore omitted. Table 4.1, on the other hand, shows the cost of our protocols in terms of bandwidth: our obfuscation mechanism is effective in reducing the bandwidth consumption of decentralized collaborative filtering. The cost associated with our obfuscated solution is about one third of that of the solution based on cleartext profiles.

Fanout	2	4	6	8	10	15	20
CT	1.8	3.0	4.4	6.5	8.2	12	14
OPRD	0.8	1.1	1.5	1.7	2.7	2.8	4.1

Table 4.1: *Bandwidth usage in kbps per node in PlanetLab*

4.8.8 Summary

While 2-DP ensures that it is impossible to extract more information than the one revealed in the user profile, our solution offers a more advantageous trade-off between accuracy and privacy. For instance, OPRD achieves an F1-Score = 0.58 with $fs = 150$ and $pf = 0.2$, while 2-DP achieves the same with $pd = 0.2$. With these parameters, the profile of 2-DP exposes more of the user's interest (*i.e.* similarity with real profile of 0.7 for 2-DP against 0.5 for OPRD) and results in higher prediction effectiveness. (*i.e.* 80% against 26% for OPRD). In addition, as the perturbation injected in the user profiles depends on similar users in term of interests, OPRD is more resilient to censorship attacks. However, OPRD does not provide as strong guarantees as 2-DP, especially against advanced active attackers or massive groups of colluders that could leverage information collected in the system to estimate the interests of users.

4.9 Conclusions

We proposed a simple and efficient mechanism to control the trade-off between privacy and recommendation quality in decentralized collaborative filtering. Our mechanism relies on two components: (*1*) an original obfuscation mechanism revealing only the least sensitive information in the profiles of users, and (*2*) a randomization-based dissemination algorithm ensuring differential privacy during the dissemination process. Interestingly, our mechanism achieves similar results as a differentially private alternative in terms of recommendation quality while protecting the sensitive information of users more efficiently and being more resilient to censorship attacks.

Chapter 5

Democratizing Personalization

The ever-growing amount of information available on the Internet can only be handled with appropriate personalization. As shown in previous chapters, one of the most appealing ways to filter content matching users' interests and achieve efficient personalized recommendation is collaborative filtering (CF). Yet, compared to a fully decentralized approach, centralized CF systems are notoriously resource greedy. Their classical implementation schemes require a substantial increase in the size of the data centers hosting the underlying computations when the number of users and the volume of information to filter increase.

This chapter explores a novel scheme and presents DeRec, *an online cost-effective scalable architecture for CF personalization. In short,* DeRec *democratizes the recommendation process by enabling content-providers to offer personalized services to their users at a minimal investment cost.* DeRec *achieves this by combining the manageability of centralized solutions with the scalability of decentralization.* DeRec *relies on a hybrid architecture consisting of a lightweight back-end manager capable of offloading CPU-intensive recommendation tasks to front-end user browsers.*

5.1 Introduction

Personalization has become an essential tool to navigate the wealth of information available on the Internet. Yet, the need to personalize content is no longer an exclusive requirement of large companies. It is arguably crucial for every web content editor, including relatively small ones. Most of these, besides publishing content (news, articles, reviews, etc), let users comment and discuss this content. This, in turn, leads to the generation of a continuous stream of information with the aim of attracting more users. Not surprisingly, this sometimes backfires when the amount of generated information becomes unmanageable to the point where many of the users eventually give up. Computer games magazines are good examples of this phenomenon as computer games is one of the fastest growing business, generating a huge activity on the Internet. The example of a relatively small French online video-game magazine (employing approximately 20 people) who gathers 3 million registered users is typical. Interestingly, these are only responsible for 20% of the traffic: the rest being generated by unregistered users. The figures are eloquent: the site of the company, visited 45 million times per month, enables access to $6,000$ discussion forums, which in turn generate up to $300,000$ messages per day. The amount of information may over-flood specialized users (interested in specific games) who finally unregister.

Similar companies would greatly benefit from a personalized interface providing users with recommendations on the media and comments they would most likely be interested in. This however represents a significant investment for small companies to buy or rent the required computing power for an effective personalization.

In this chapter, we present and extensively evaluate *DeRec*, a novel architecture capable of providing a cost-effective personalization platform to web content editors. Instead of scaling through larger and larger recommendation back-ends, *DeRec* delegates expensive computation tasks to front-end web browsers running on client machines, while, at the same time, retaining the system's coordination on the server side.

DeRec implements a *user-based collaborative filtering* (CF) pattern. Its user-based variant represents a natural opportunity for decentralizing recommendation tasks on users' machines, where each user is herself in charge of the computation of her personalization operations. *DeRec* adopts a k-nearest neighbor strategy for it is more natural to distribute than alternative ones (such as matrix factorization). The idea consists in computing the k-nearest neighbors according to a similarity metric, and identifying the items to recommend from this set of neighbors [135]. The challenge is to cope with a large number of users and items. Traditional recommendation architectures achieve this by computing neighborhood information offline and exploiting elastic cloud platforms to massively parallelize the recommendation jobs on a large numbers of nodes [52, 56]. Yet, offline computation is less effective when new content is being added continuously. This makes real-time recommendations very hard to achieve and forces the periodic re-computation of predictions, inducing significant running costs [52, 102] and power consumption [108].

DeRec's hybrid architecture avoids the need to process the entire sets of users and items at the same time by means of a sampling-based approach inspired from epidemic computing [32, 143]. The *DeRec* back-end server provides each front-end user's web browser with a sample set of other users. The browser then computes its user's k nearest neighbors and most popular items based on this sample. The process continues with the server's using the user's new neighbors to compute the next sample. This leads to a feedback mechanism that improves the quality of the selected neighbors and leads them to converge very quickly to those that could have been computed using global knowledge. While this involves extra communication between the server and the clients, there is a clear advantage in terms of computational cost as demonstrated by our evaluations.

DeRec is generic and applicable to any CF system that processes the user-item matrix in a user-centric fashion. Content providers can customize *DeRec* with a specific sampling procedure or an application-tailored similarity metric. *DeRec* also incorporates an adaptation mechanism that can tune the computational tasks on both the server and the clients according to their capability footprints. *DeRec* can accommodate client machines ranging from large desktop computers to smaller mobile devices. It also allows the server to operate effectively even during load peaks. The architecture of *DeRec* is hybrid in the sense that it lies between traditional centralized systems and fully decentralized solutions. such as [126] and [109]. In this respect, *DeRec* provides the scalability of decentralization without forcing content providers to give up the control of the system. Unlike fully decentralized approaches, its lightweight web widget does not require clients to install specific software, and its centralization of system aspects, like connections and disconnections to and from the system, enables its realistic deployment in a dynamic system.

We extensively evaluate *DeRec* in the context of two use cases, a personalized feed, Digg, and a movie recommender, MovieLens, using real traces in both cases. We compare *DeRec* against solutions based on a centralized infrastructure with an offline neighbor-selection process and an online item-recommendation step. Our results show that *DeRec* reduces the server's computational requirements by a factor ranging from 2.1 to 8.5 while preserving the quality of recommendation and with only limited computational and bandwidth costs for client machines. We show that, as the scale of the system increases, the load on the server grows much more slowly with *DeRec* than with a centralized approach. We also show that *DeRec* successfully adapts to various client capabilities by reducing the size of the sample provided by the server. This demonstrates the viability of *DeRec* as an alternative to data-center-based approaches by leveraging the computational power of clients. We believe that *DeRec* can be adopted by any content editor wishing to provide personalized recommendations to their users at a low cost.

Contributions

To summarize, this chapter explores a novel hybrid architecture for personalizing recommendations by combining the scalability and cost-effectiveness of massively distributed systems with the ease of management of centralized solutions.

The rest of the chapter is organized as follows: Section 5.2 discusses related work. Section 5.3 discusses some background. Section 5.4 describes in details the different components of *DeRec*. Section 5.5 provides some information on the implementation while the extensive evaluation of *DeRec* is presented in Section 5.6. Finally, Section 5.7 concludes the chapter.

5.2 Related Work

One of the main challenges underlying collaborative filtering is scalability. To address this challenge, centralized approaches leverage massive parallelization through map-reduce jobs on expensive elastic cloud architectures [41,52,114]. A more radical way to address scalability is through a significant departure from centralized (or cloud-based) architectures, namely through fully distributed CF solutions in peer-to-peer settings as we present in previous chapters.

Whereas very elegant and appealing in theory, the requirement to have a piece of software on every client, the need for synchronization between multiple devices, and the management of users' online and offline patterns make these solutions practically hard to deploy.

Nonetheless, their inherent scalability provides a strong motivation for a hybrid approach like ours which separates the concerns: a centralized back-end handles the connections and disconnections of users whereas front-ends perform the actual personalization on the client side.

A similar idea was explored by TiVo [25] in the context of an item-based CF system. Yet, TiVo does not completely decentralize the personalization process. It only offloads the computation of item-recommendation scores to clients, while it computes the correlations between items on the server side. Since the latter operation is extremely expensive, TiVo's server only computes new correlations every two weeks, while its clients identify new recommendations once a day. This makes TiVo unsuitable for dynamic websites dealing in real time with continuous streams of items. In contrast, *DeRec* addresses this limitation by delegating the entire filtering process to clients: it is to our knowledge the first system to do so and it is applicable to any user-based CF platform.

5.3 Overview and Background

Our personalized recommendation is similar to the one presented in the previous chapters. It can be stated as follows. Consider a set of users $U = u_1, u_2, ..., u_N$ and a set of items $I = i_1, i_2, ..., i_M$, Each user $u \in U$ has a profile P_u collecting her opinions on the items she has been exposed to. This consists of a set of quadruplets $P_u = < u, i, v, t >$ where u is a user, i an item, and v the score value representing the opinion of user u on i as recorded at time t. For the sake of simplicity, we only consider binary ratings indicating whether a user *liked* or *disliked* an item after being exposed to it. [1]

The goal is to provide each user with a set of items $R \subseteq I$ which she is likely to appreciate. We consider a user-based CF using a sampling-based approach inspired from epidemics, with minimal computational cost on the server side. We achieve this by exploiting the computational resources available at the user, via the browsers of online users, in contrast to a centralized architecture which achieves the recommendation tasks on the server side. Before detailing this, we provide some background on user-based CF.

5.3.1 User-Based CF

A user-based CF system operates in two steps: *neighbor selection* and *item recommendation*. Neighbor selection consists in determining, for each user u, a set of similar users with respect to a given similarity metric. These neighbors are assumed to be the users that most closely match the user's interests. Several metrics can be considered. For the sake of simplicity, we consider here the well-known cosine similarity metric [63], computing the cosine of the angle formed by two users profiles $\sigma(P_u, P_n)$ where $u, n \in U$.[2] To avoid computing the similarity of a user with all other users, we consider a sampling-based approach to reduce the dimension of the problem. The CF system then uses the selected neighbors to recommend items to a user that she has not yet been exposed to. In this work, we consider the top-r most popular items of the extended neighborhood as in [52].

Both the computation of neighborhood and item recommendation are solely based on the content of user profiles. As many systems need to react immediately to new user requirements and potential changes of interests, recommendations should not take a user's entire rating history. A sliding time window, called *profile window*, is used to select which user ratings to consider for the recommendation. Depending on the dynamics of the application, the size of the profile window varies from a few hours to several months or may include only the x last ratings. Here, we consider only the opinions expressed by users on the last T_{win} items, where T_{win} is a system parameter. [3]

[1]This rating can be easily extended to multi ratings.

[2]Note that *DeRec* can easily be parametrized with other similarity metrics (*e.g.* Pearson correlation, probability based, Jaccard, etc).

[3]In CF systems, the presence of new users (*i.e* with empty profiles) and new items (*i.e.* without

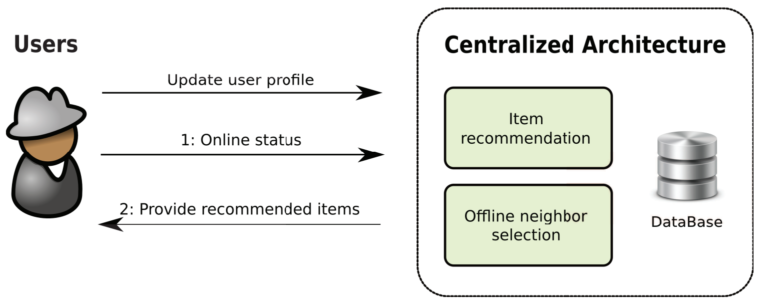


Figure 5.1: *Centralized Architecture (offline recommendation).*

5.3.2 A reference centralized architecture

The typical architecture of a CF system for web applications follows a client-server interaction model similar to the one depicted in Figure 5.1. Users interact with a web browser, which in turn communicates with the web server of the application providing it with information about the user's interests (*e.g.* clicks or ratings) and receiving the results of the recommendation process.

In this work, we consider the architecture described in [52] to compare *DeRec* to. The content provider stores the information received from clients on a database, and performs the tasks identified above. First, it computes and updates the neighborhood information. Due to its high computational cost, this task is carried out offline, in a periodical fashion, according to the dynamics of items. Second, the server determines the items that should be recommended to each user by selecting the r most popular items in the user's extended neighborhood. This item-recommendation task is performed in real time.

In either case, the main characteristic of this architecture is that all computation-intensive tasks are performed at the server. The web server, application logic, recommendation subsystem, and associated databases may leverage distribution, for example using cloud-based solutions. Yet, all active components remain under the responsibility of the website owner. For instance, Google News [52] employs a cloud-based solution and delegates computation- and data-intensive recommendation tasks to a data center. This allows its recommendation system to scale to large numbers of users and items, but requires significant investments to provide recommendations within satisfactory response times. This makes such server-based architectures viable only for large companies that are able to sustain the associated costs.

ratings) leads to the so-called *cold-start* issue. This, however, is application dependent. As our system can be easily parametrized to address this problem as needed, the cold-start evaluation is out of the scope of this work.

5.4 DeRec

DeRec addresses the limitations of existing architectures by providing personalized recommendations without incurring the significant infrastructure costs associated with traditional recommendation solutions for web content providers. *DeRec* achieves scalability by decentralizing the computation of both the intensive tasks of the recommendation process on the browsers of online users, while maintaining a centralized system orchestration. This makes it possible to exploit connected users for scalability purposes in a transparent manner, without suffering from the limitations that characterize fully decentralized architectures such as P2P ones (*i.e.* the need to install a specific client application and the dynamic behavior of users).

The high-level architecture of *DeRec* is depicted in Figure 5.2. Similar to existing systems, users' web browsers interact with a front-end server that maintains the user-profile database and provides the actual recommended items once they have been identified by the recommendation process. However, the computationally intensive tasks of item recommendation and neighbor selection are offloaded to users and executed within their browsers. To achieve this goal, the *DeRec* server uses a sampling-based approach to efficiently split and dispatch personalization jobs to users. This approach limits network traffic by constraining the size of the set from which neighbors are selected while preserving the quality of the recommendation. The *DeRec* widget, in turn, executes neighbor selection and item recommendation through Javascript code. This makes the decentralization transparent to users and leverages users' browser as part of the *DeRec* framework.

In the following, we first describe the basic operations of the *DeRec* server by describing how it distributes the computation of the two main steps of user-based CF. Then, we present the *DeRec* widget's operations and its interactions with the server, before introducing the footprint capability adaptation. This mechanism allows *DeRec* to dynamically adapt the size of the computation jobs sent to users according to the capability of their associated devices and the load of the server. More details about how *DeRec* can be tuned by content providers are provided in Section 5.5.

5.4.1 *DeRec* server

The key feature of the *DeRec* server is the ability to decentralize the computationally intensive tasks of user-based CF described above. The recommendation cost is thus shared between the server and a large number of online users instead of being sustained by the server only. The server directs personalization jobs (comprising item recommendation and neighbor selection) to online users by means of the two main components depicted in Figure 5.2: the sampler and the job dispatcher.

In order to reduce traffic, *DeRec* does not require each user to select her k nearest neighbors from the entire database. Instead, the server uses the sampler to provide each user with a sample of candidates with respect to which the user should compute her similarity in order to update her neighbors and select her recommended items.

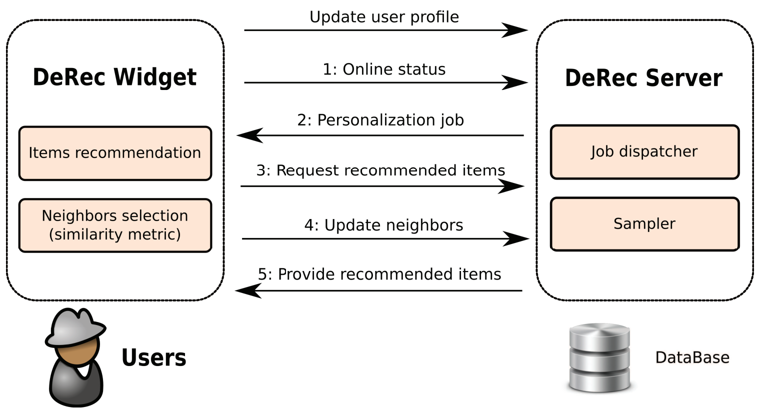


Figure 5.2: *Main tasks and components of* DeRec*.*

This sampling-based approach allows *DeRec* to reduce the size of the recommendation problem.

The job dispatcher packs each prepared sample into a personalization job (essentially a message containing the sample) to be executed by online users, and then collects the results of their computations. Consequently, the entire recommendation process in *DeRec* from the content provider's point of view is reduced to selecting a sample and to preparing, sending and collecting the results of the personalization job for the online users only. We describe in details the sampler and the job dispatcher in the following.

Sampler

The sampler is solicited by the job dispatcher to generate a sample of candidate users S with respect to which a user should compute its similarity. The sampler builds such a sample $S_u(t)$ for user u at time t by aggregating three sets: (i) the current neighbors N_u of u, (ii) their neighbors, and (iii) a random set of other users. Let k be a system parameter determining the size of a user's neighborhood, N_u. Then the sample contains at most k one-hop neighbors, k^2 two-hop neighbors, and k random users. Because these sets may contain duplicate entries, the size of the sample is $\leq 2k + k^2$. The job dispatcher can further reduce this size to take into account the capability footprints of both the user and the server as explained in Section 5.4.3.

The periodic computation of samples takes its inspiration from epidemic clustering protocols [32, 143]. Using u's neighbors and their neighbors provides the widget with a set of candidates that are likely to have a high similarity with u. Adding random users to the sample prevents this search from getting stuck into a local optimum and guarantees that the process will eventually converge by recording the user's k-nearest neighbors in the set N_u, so that $\lim_{t\to\infty} N_u - N_u^* = 0$, where N_u^* is the optimal set (*i.e.* containing the k most similar users). Research on epidemic protocols [143] has shown that convergence is achieved very rapidly even in very large networks.

Job dispatcher

The job dispatcher manages the distribution of these personalization jobs to users. Once it receives an online notification from a user u (arrow 1 in Figure 5.2), it asks the sampler for the subset of candidates described above, the sample. In doing so, it asks the sampler for an adjusted sample size according to the capability footprints as explained in Section 5.4.3. Then, it prepares the personalization job for u by building a message including its profile and the profiles of all candidates returned by the sampler (arrow 2 in Figure 5.2). Finally, it manages the interaction with the *DeRec* widget. It sends the personalization job to it, and collects the results of neighbor selection and item recommendation. First, it stores the former into the database. Then, it processes the latter by sending the widget the actual content of the selected items (arrow 5 in Figure 5.2).

5.4.2 *DeRec* widget

Similar to existing systems, users interact with *DeRec* through a web interface that provides them with personalized feeds of items. In *DeRec*, this consists of a widget written in Javascript. The widget acts as web container and interacts with the *DeRec* server using a web API. Javascript has been widely adopted and makes it possible to create dynamic web interfaces, for example by proactively refreshing their content. *DeRec*'s widget leverages this technology to massively distribute the main tasks of recommendation to the browsers of online users. In the following, we describe the widget's operation, while we provide additional implementation details in Section 5.5.

The *DeRec* widget sits on the client side and manages all the interactions with the server side. Consider a user u. First, the widget is responsible for updating the user profile stored on the server. To achieve this, it contacts the server whenever u expresses an opinion on an item (*update-profile* arrow in Figure 5.2). Second, the widget is responsible for refreshing the displayed recommendations. To achieve this, it periodically contacts the server with an *online-status* message indicating that the user is online (Arrow 1 in Figure 5.2). The server replies to this message by sending a personalization job containing a sample of users along with their associated profiles (Arrow 2). Upon receiving this job, the widget computes u's personalized recommendations as $R_u = \alpha(S_u, P_u)$, where $\alpha(S_u, P_u)$ returns the identifiers of the r most popular items among those that appear in the profiles in S_u, but not in P_u. These are the most popular items in the sample to which u has not yet been exposed. The widget then requests the actual content of these items by sending the selected identifiers to the server (Arrow 3 in Figure 5.2). When the server replies to this request (Arrow 5 in Figure 5.2), the widget displays the items to the user.

It is worth remembering that the sample, S_u, contains mostly users that are in u's two-hop neighborhood, together with a small number of randomly selected users. By taking into account the items appreciated by the former, the widget exploits the opinions of similar users. By also taking into account those appreciated by the latter, it also includes some popular items that may improve the serendipity of its recommendations.

After requesting the recommended items from the server, the widget also proceeds to updating the user's k-nearest neighbors. To achieve this, it computes the similarity between u's profile and each of the profiles of the users in the sample. It then retains the users that exhibit the highest similarity values as u's new neighbors, $N_u = \gamma(P_u, S_u)$, where $\gamma(P_u, S_u)$ denotes the k users from S_u whose profiles are most similar to P_u. It then returns these neighbors to the server to update the database (arrow 4 in Figure 5.2). The server will use these new neighbors the next time it has to compute a new sample.

5.4.3 *DeRec* capability footprint adaptation

More and more users use smart phones to browse their information feeds or to connect to social applications. To account for the heterogeneity of user devices in term of computation power, *DeRec* provides a device-capability adaptation mechanism. More specifically, the widget periodically computes a capability footprint of its associated device and sends it to the server. This capability footprint measures the time spent by the device to solve a computational puzzle (*i.e.* reversing an MD5). The *DeRec* server keeps track of this capability footprint for each online user. Since, there is a direct correlation between the size of the sample and the cost of the personalization job, the *job dispatcher* uses the footprint capability to adapt the size of the *sample* that constitutes the candidate list for the personalization jobs. To make this possible, content providers can define a function that maps each capability-footprint value onto a percentage p_{cap}. The job dispatcher then asks the sampler for a sample consisting of $kp_{\text{cap}}/100$ one-hop neighbors, $(kp_{\text{cap}}/100)^2$ two-hop neighbors, and $kp_{\text{cap}}/100$ random nodes. We evaluate the effectiveness of this mechanism in Section 5.6.

While clients may be limited depending on their device capabilities, the server might also experience picks in load that limit its operation. This is reflected in the capability footprint attached to the server, which is directly correlated to the number of connected users. *DeRec* also accounts for such limitations by adapting the server's operations according to its footprint. Similarly to the clients, the server uses an internal footprint to adapt the size of the samples that constitutes candidate list. As for the clients, the adpated size is expressed as a percentage of the default sample, parametrized by k.

5.5 Implementation

Our implementation of *DeRec* consists of a set of server-side modules and a client-side widget following the architecture described in Section 5.4. Each component of the server consists of a J2EE servlet. These servlets come in two flavors: either bundled all together with a version of Jetty [6], a lightweight web-server, or as a

`https://`*DeRec*`/profile/?uid=uid&rid=rid&like=[0:1]`	Update user profile with an opinion
`https://`*DeRec*`/online/?uid=uid&footprint=[40:100]`	Send an online status and set device footprint capability
`https://`*DeRec*`/neighbors/?uid=uid&id0=fid0&id1=fid1&...`	Update neighbor information with specified list
`https://`*DeRec*`/itemRec/?uid=uid&rid0=id0&rid1=id1&...`	Request the specified items
`https://`*DeRec*`/logout/?uid=uid`	Send a logout notification

Table 5.1: *Web API to custom* DeRec*.*

`Interface SetSampler();`	Define the sampling strategy
`Interface SetProfile();`	Define the user profile sent in selection jobs
`Interface SetSimilarity();`	Determine the similarity metric for the neighbors selection
`Interface SetRecommendedItems();`	Determine the selected items to recommend

Table 5.2: *Interfaces to custom* DeRec*.*

stand-alone component that can be run in a web server. Integrating all components into a customized web server allows content providers to deploy our solution into their existing web architectures. Moreover, bundling each component with a Jetty instance makes it easy to deploy our architecture on multiple hosts thereby balancing the load associated with the various recommendation tasks (*e.g.* network load balancing).

The *DeRec* client consists of a web widget. A widget is a piece of Javascript acting as a web container that can easily be embedded in third-party sites, or online and offline dashboards (*i.e.* netvibes, igoogle, web interface). This Javascript defines the behavior of the widget: it collects user opinions, it executes the personalization jobs (neighbor selection and the item recommendation), it receives recommendations, and it displays them. To do so, it communicates with the *DeRec* server through the Web API described in Table 5.1. The use of a public Web API not only provides a simple way to implement our widget but also achieves authentication and makes it possible for content providers and even users to build their own widgets that interact with the framework. To develop a new widget, one simply needs to make the right API calls and import the Javascript file associated with *DeRec* widget that deals with processing selection jobs. All exchanges from the server to the widgets of online users are formatted in JSON. We use the Jackson implementation [5], one of the fastest solutions to serialize JAVA objects to JSON message.

To parametrize *DeRec*, content providers can specify a specific similarity metric or item recommendation algorithm in the Javascript file which defines the behavior of the widget, and includes the desired fields of the user profile in the JSON messages. The current version of *DeRec* integrates interfaces to easily customize parts of its behavior (Table 5.2).

5.6 Evaluation

We extensively evaluate *DeRec* through simulation on real traces from Digg and Movielens along the following metrics: (*i*) the time to perform all operations on both the *DeRec* server and the *DeRec* widget. This measures the ability of *DeRec* to reduce the load on the server and offload tasks to clients; (*ii*) the impact of the profile windows; (*iii*) the behavior of *DeRec* with a growing number of online users; (*iv*) the benefit of *DeRec*'s footprint capability adaptation to balance the load on the server and the clients; (*v*) the communication overhead; and (*vi*) the quality of the provided recommendation. In addition, we compare *DeRec* against fully centralized candidates for both neighbor selection and item recommendation.

5.6.1 Experimental setup

Platform

In our experiments, we consider a single server hosting all the components of the architecture (front and back-end) and we assume that the database is entirely stored in memory. Obviously in practice, several machines can be used to implement each component separately to sustain the load at the network level. However, as this load balancing technique does not affect the outcome of our experiments, its evaluation is out of scope in this work. In our experiments, we use two PowerEdges 2950 III, Bi Quad Core 2.5GHz, with 32 GB of memory and Gigabit Ethernet, respectively to evaluate the server and the clients. In addition, we rely on Jetty [6] to embed a web server and messages are formatted in JSON with the Jackson implementation [5].

Datasets

Our experiments use real traces from both a movie recommender based on the MovieLens workload [10] and Digg [2], a social news website. The MovieLens dataset consists of movie-rating data collected through the MovieLens recommender web site during a seven-month period (Sept. 1997-April 1998). The dataset is composed of the users who rated more than 20 movies. For the sake of simplicity, we project the ratings into a binary rating as follows: for each item in a user profile, the rating is set to 1 if the initial rating of the user for that movie is above the average rating of the users across all her items, 0 otherwise. The Movielens dataset (ML1) allows us to compare the quality of the recommendation provided by *DeRec* against the one of the Google news personalization reported in [52] which relies on the very same dataset. In addition, since we use three sizes of this dataset, this enables us to assess how *DeRec* scales up when the number of users increases. Table 5.3 summarizes the workload figures.

We also use a Digg dataset to study a situation with a highly dynamic feed of items. Digg is a social news website to discover and share content where the value of a piece of news is collectively determined. We collected traces from Digg for approximately

Dataset	Users	Items	Ratings
MovieLens1 (ML1)	943	1,700 movies	100,000
MovieLens2 (ML2)	6,040	4,000 movies	1,000,000
MovieLens3 (ML3)	69,878	10,000 movies	10,000,000
Digg	59,167	7,724 items	782,807

Table 5.3: Datasets statistics

$60,000$ users and more than $7,500$ news over 2 weeks in 2010. Figures are reported in Table 5.3. This dataset contains all observed users in the specified period without any threshold on the number of ratings.

Online patterns

In *DeRec*, a recommendation is computed for a user when she is online. In order to evaluate the impact of the number of online users on *DeRec*, we consider several online patterns. To this end, we use the timestamp attached to user ratings in the datasets. We split the trace in timeslots and select the online users for each slot. A user is considered online if she provided at least one rating during a slot. Moreover, as users can be connected to *DeRec* without providing ratings (*i.e.* reading recommendations), we artificially add online users. We define an online pattern as a percentage of additional random users added to the ones which provided rating at each time slot. An online pattern of 0% means that only users which provided a rating at the associated time slot are considered.

Methodology

In order to evaluate *DeRec*, we run experiments simulating its operations over time by replaying the activity and ratings of users. In each slot, every online user sends an online notification to the server and in turn receives a sample from the *DeRec* server and uses it to perform the personalization tasks. Upon completion of those tasks, each client *(i)* sends a request to the server to get the recommended items and *(ii)* send an update of its k nearest neighbors. The server then provides the desired items to users and updates the profile database. To simulate real exchanges, each item provided by the server contains real content from a RSS feed item of 1004 bytes. Users also send their profile updates to inform the server about the items they liked or disliked for each rating present in the dataset during the associated slot. As described in Section 5.4.3, the job dispatcher tunes the size of requested samples according to footprint capabilities. To model this, as well as the mapping of capabilities onto percentages we associate each user with a random value between 40 and 100. This represents the percentage of the neighborhood size, k, taken into account by the sampler. For simplicity, we abuse terminology and refer to this value as capability footprint. A user u with a 50% footprint in a system with $k = 10$, will receive a sample containing its 5 closest neighbors, the 5 closest neighbors of each such neighbor, and 5 random users, thus receiving a sample of size 35. Unless specified otherwise, we set the capability footprint for the *DeRec* server to 100%. Finally, users send a logout notification at the end of the experiment. Default parameter values used in our experiments are summarized in Table 5.4.

Parameter	Value
Size of the neighborhood	10
Sample size	≤ 120
Time slot (MovieLens)	12 hours
Time slot (Digg)	1 hour
Recommended items	10 items
Online pattern	5%
Server capability footprint (server)	100%
Device capability footprint (clients)	[40-100%]
Windows profile	100 items
Offline clustering period (MovieLens)	48 hours
Offline clustering period (Digg)	24 hours

Table 5.4: Default parameter setting

Metrics

We measure the time spent on both the *DeRec* server and the *DeRec* widget to achieve their operations. The reported time includes the time needed to receive and to send the packets from or to the server and the widgets. In addition, we measure the bandwidth consumption between the *DeRec* server and the *DeRec* widgets. Regarding the evaluation of the recommendation quality, similar to most machine learning evaluation methodologies, we split the datasets in sub training and test sets (80% training set - 20% testing set). We report both the precision and recall for users according to the number of items provided to users as mentioned in Section 4.7.3. In short, precision reflects the accuracy of the recommendation defined as the number of interesting items received over the total number of received items. On the other hand, the recall expresses its completeness as the number of interested items received over the number of items which should be received. In order to compare our solution with previous works [52], we did not use windows profile and considered all users ratings for the evaluation of the recommendation. Similarly, for the sake of comparison and to standardize the results, we mainly report them of the smallest MovieLens dataset (*i.e.* ML1).

5.6.2 Baseline for comparison

We compare *DeRec* against the centralized recommender solution depicted in Figure 5.1. In order to ensure a fair comparison, we select several alternative approaches and use the least expensive as a baseline in the rest of the experiments. The alternative approaches differ only in their k-neighbor selection algorithm, and use the same client-server protocol as *DeRec*. Therefore, we consider the same metrics for all aternatives (*i.e.* time, bandwidth consumption and quality of the recommendation).

k-neighbor selection

In a centralized architecture, the neighbor selection task is achieved periodically offline by the server. More precisely, we define an offline clustering period of 48 hours and 24 hours for the MovieLens and Digg datasets respectively, as reported in Table 5.4. We consider several alternatives to select the k-nearest neighbors of each user: *(i)* an exhaustive and multi-threaded approach computing the similarity between a user and all other users in the system; *(ii)* a solution provided by Mahout, an open-source machine-learning Apache library [9]; *(iii)* a probabilistic approach using Minhash similar to [52] available in [8], relying on a clustering model which performs dimension reduction of the problem by assigning users to the same cluster according to the probability of the overlap between the set of items they liked in common; *(iv)* a multi-threaded solution, called *CRec*, using the same algorithm as *DeRec* (*i.e.* using a sampling approach) but executed in a centralized manner.

We evaluate the total amount of time spent to achieve the k-neighbor selection task on all centralized candidates. Their parameters are set to the same values as in *DeRec*. More precisely the profile window is limited to the last 100 rated items. To simulate a parallel computation on a 10 node cluster of both the Mahout and Minhash approaches, we report a lower bound of the computation time *(i.e.* divided by 10). The results are depicted in Figure 5.3. We observe that *CRec* consistently outperforms other approaches. Therefore, we select *CRec* to compare *DeRec* with in the rest of this chapter. Note that the quality of the recommendation provided by all these approaches is similar (as shown in Section 5.6.8).

Item recommendation

In the centralized architecture, the same recommendation solution as for *DeRec* is used. More precisely, as described in Section 5.4.2, every time a user sends an online status to get a recommendation, the server computes the most popular items among her extended neighborhood (*i.e.* group composed of its direct neighbors and the neighbors of these direct neighbors and random users) and then provides to the client the $r$most popular items unknown to the user.

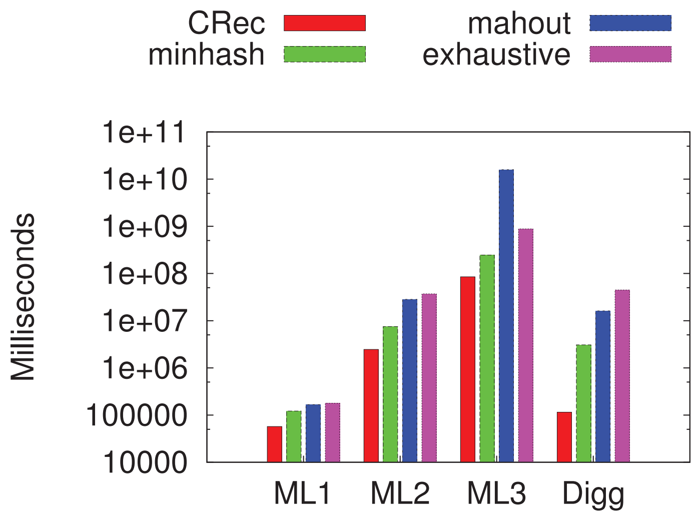


Figure 5.3: Time to compute the k nearest neighbors.

5.6.3 *DeRec* versus *CRec*

We now compare *DeRec* against *CRec*, representative of the centralized approach.

Server evaluation

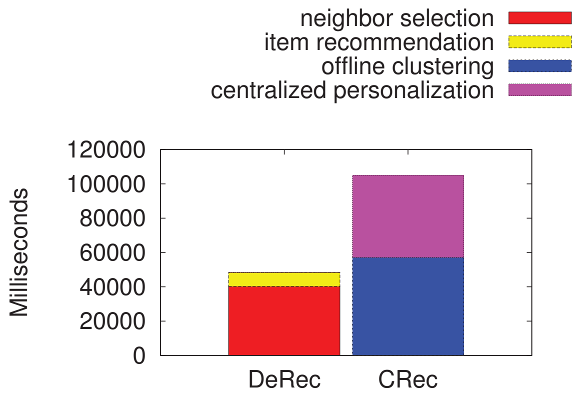
Table 5.5 presents the breakdown of computation times among the different operations managed by the *DeRec* server for the ML1 MovieLens dataset. Results show that the most time-consuming task in *DeRec* server is the one forming and managing the JSON messages to send the samples to users. Other operations such as the sampling itself, or the management of neighborhood and profile updates, recommendation requests, and logout notifications are negligible.

Task	Time (ms)	%
Profile updates	1,297	2.6
Logout	12	0.02
Sampling	1,226	2.4
Build the sample message	38,200	**76.8**
Neighborhood updates	842	1.6
Recommendation requests	4,259	8.5
Build the recommended items message	3,874	7.7
Total	49,710	100

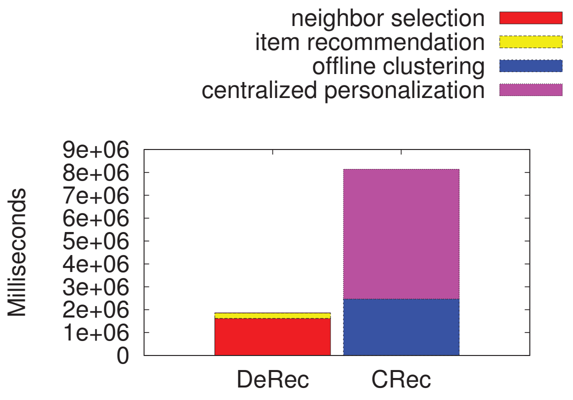
Table 5.5: *DeRec* server operations for ML1.

Figure 5.4 and Table 5.6 depict the comparison of *DeRec* against *CRec*. We show that the total amount of time consumed on the server is drastically reduced in

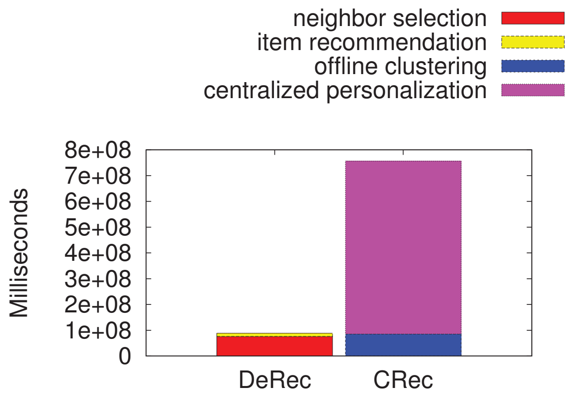
DeRec compared to *CRec*. Indeed, in *DeRec* the message management dominates over computing time and consumes much less time, on average 25% for the neighbor selection on MovieLens datasets, and 92% for the item recommendation on all datasets. On the Digg dataset, *DeRec* spends more time for the neighbor selection task than *CRec* for the offline clustering. This is mainly due to the small size of the users profiles on the Digg dataset (on average 13 for Digg compared to 138 for MovieLens). Indeed, the computation time of offline clustering depends on the size of user profiles, in contrast to *DeRec* (server) in which this is restricted to message management. In addition, the cost of offline clustering is correlated with its frequency: the more frequent it is, the longer it requires. In ML1, if we consider *CRec* with an offline clustering period that is twice as fast (*i.e.* 24 hours), *DeRec*'s improvement increases from 30% to 65%.



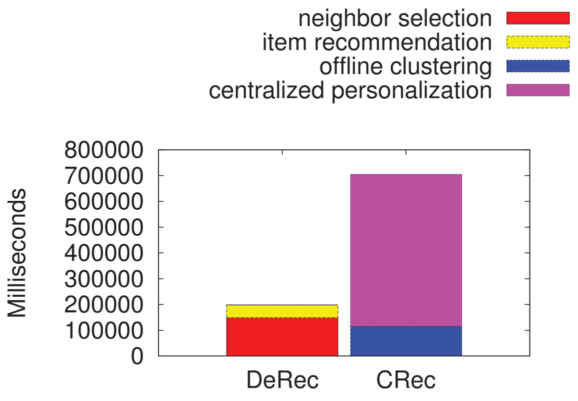
(a) ML1



(b) ML2



(c) ML3



(d) Digg

Figure 5.4: *DeRec* server versus *CRec* (all datasets).

Moreover, the gap between *DeRec* and *CRec* increases according to the number of users. The server in *DeRec* sees its load increase two times more slowly than *CRec* with a 6-fold scale increase in the number of users (*i.e.* ML1 to ML2), and four

Component	ML1	ML2	ML3
DeRec server	49,710	1,859,213	88,609,095
All *DeRec* widgets	219,642	7,092,467	336,504,783
CRec	104,812	8,138,192	756,289,396

Table 5.6: *DeRec* server operations for ML1.

times more slowly with a 70-fold scale increase (*i.e.* ML1 to ML3). While the total computation time in all *DeRec* widgets is larger than the time spent by *CRec* for the entire recommendation process of ML1, from a content provider perspective, the load on its infrastructure is reduced by a factor of 2.1.

DeRec widget evaluation

We now evaluate the cost of operating *DeRec* on the client. The new action introduced on the client by our solution compared to the centralized one, is the management of personalization jobs including the item recommendation, the k-nearest-neighbor computation and the update messages sent to the server.

We measure the time needed by the widget to achieve these different tasks on Table 5.7. We observe that about 50% of the time is spent on the item recommendation and the neighbor selection, the other 50% being shared by the request and the reception of the recommended items, the profile and the neighborhood updates. Similar to the *DeRec* server, the message management dominates over computing time even on the *DeRec* widget which limits the computation capability required on the client.

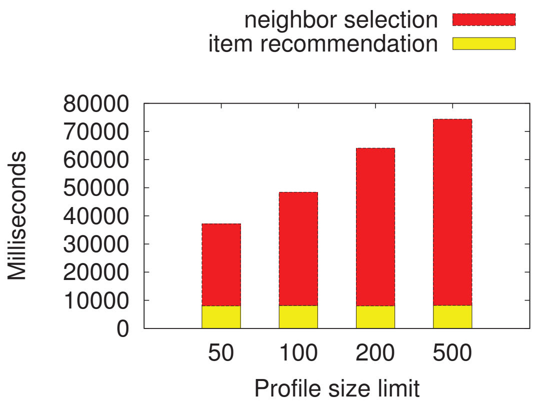
Finally, we measured the time spent by the widget within a browser (*i.e.* Firefox) to perform the selection jobs. An average of 911 milliseconds is spent by the browser to select the k nearest neighbors from the received sample and to identify the most popular items in its extended neighborhood. This task takes only 5.6 times longer than the time required by the widget to get and display an RSS feed from Digg, which makes *DeRec* clearly acceptable for users. In addition, this task is entirely transparent to them thanks to the asynchronous communication of the AJAX model.

Task	Time (ms)	%
Profile updates	22.16	17.2
Logout	0.19	0.1
Neighborhood updates	10.79	8.4
Neighbor selection & items recommendation	67.90	**52.9**
Requests and receives desired items	27.12	**21.1**
Total	128.16	100

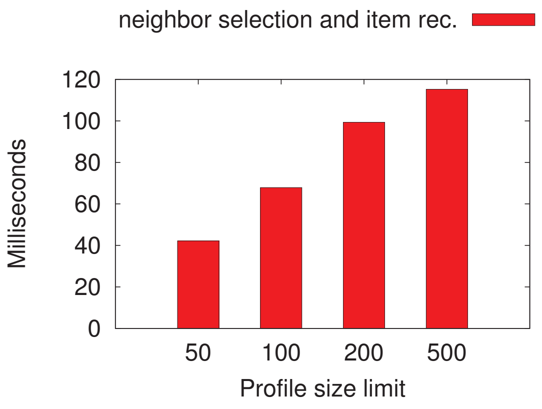
Table 5.7: *DeRec* widget operations for ML1.

5.6.4 Impact of the profile-window size

The size of the user profile directly impacts the performance of *DeRec*. This largely depends on the dynamics of applications: typically users tend to rate much more often news than movies. More precisely, the larger the profile, the larger the size of the JSON messages generated by the server. The windows size directly impacts *DeRec* since it increases or decreases the time spent by the *DeRec* server to build up JSON messages. Figure 5.5 shows that the time required to prepare selection jobs increases only by a factor of 2 when the profile size changes from 50 to 500. Obviously, the time spent by the server to provide recommendation does not change according to the profile window (5.5a). However, on the *DeRec* widget, the time increases according to the size of the profile (5.5b).



(a) *DeRec* Server



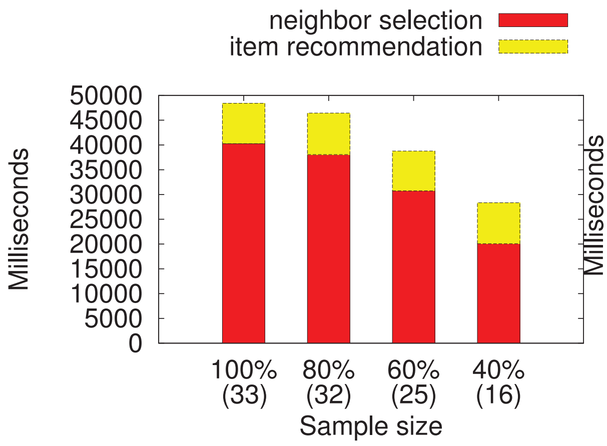
(b) *DeRec* Widget

Figure 5.5: Impact of the profile window size for ML1.

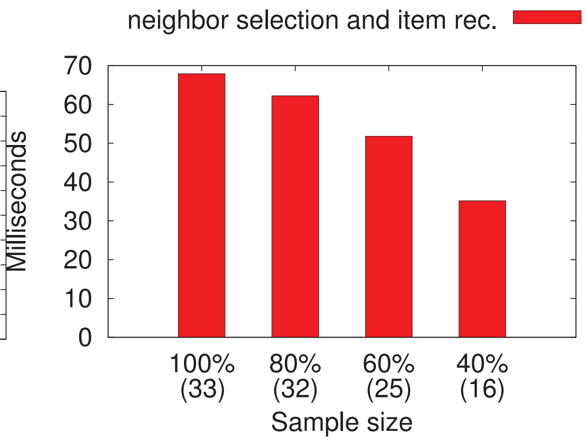
5.6.5 Capability footprint adaptation

In this section, we analyze *DeRec*'s ability to account for the heterogeneous capabilities of user devices and to balance the load on the server at the application level. We measure the time spent to achieve the different actions on both the server and a widget according to its load and its device capability, respectively. Figure 5.6 shows that by adapting the size of the sample sent to the widget, the server can reduce the time spent to form the sample message up to 45% for a server capability footprint of 100% vs 40%. On the other hand, the widget can reduce the required computation up to 48.2% from a device capability footprint from 100% to 40%.

Interestingly enough, due to the homophily characteristic of social networks [99] (*i.e.* users are more likely to be friends if they share common interests), the average sample size is notably smaller than the maximum possible sample size (*i.e.* assuming that all neighbors of neighbors are distinct) Without any adaptation, the average



(a) *DeRec* Server



(b) *DeRec* Widget

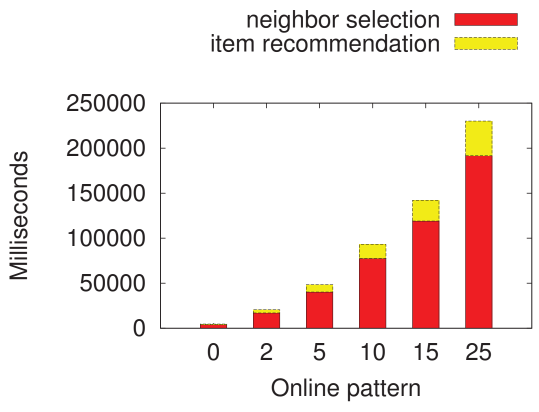
Figure 5.6: Impact of the capability footprint for ML1.

sample size is equal to 33 against an upper bound of 120 (*i.e.* sum of the 10 random users and 10 neighbors of every of the direct 10 neighbors).

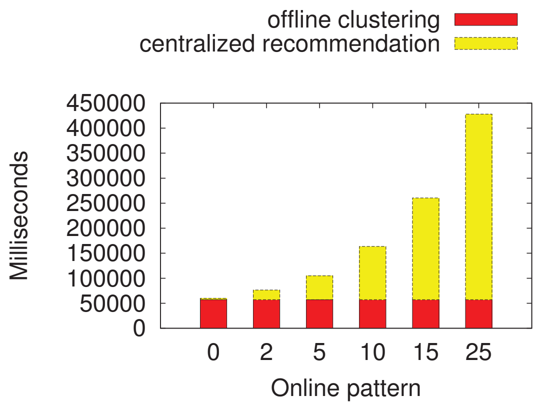
5.6.6 Online patterns

The load of the server can drastically change depending on the number of users online at the same time. Figure 5.7 compares the behavior of *DeRec* with a centralized architecture when facing a growing number of online users. In *DeRec* (5.7a), the main impact of a growing number of connected users is to generate more jobs to form and to send to online users. In contrast, in the centralized architecture (5.7b), the offline clustering takes the same time regardless of the number of clients. However, the time spent to achieve the item recommendation grows exponentially according to the online pattern as the server computes item recommendation for all online users.

However, unlike the *DeRec* server, the *DeRec* widget is not impacted by a growing number of users: each user is in charge of her own computation on a given sample provided by the server and the size of which does not vary with the size of the system. Figure 5.8 shows that for an online pattern from 2 to 25, the average time spent by users remains around 4 milliseconds for the MovieLens datasets and around 0 for the Digg one. This time varies according to the average size of the users profiles, The larger the profiles, the longer the computation time. An online pattern of 0 means that only users providing ratings in a time slot were defined as online in this slot, in contrast to a positive online pattern which includes additional random nodes as online to generate artificially more activity. It is well known that in social platforms, the most active users in term of rating are connected more often than the others. As a consequence, in this case, online users are mainly the ones with the largest profiles and requires more time to compute the similarity. This explains the gap between



(a) *DeRec*



(b) Centralized *DeRec*

Figure 5.7: Server with varying online patterns for ML1.

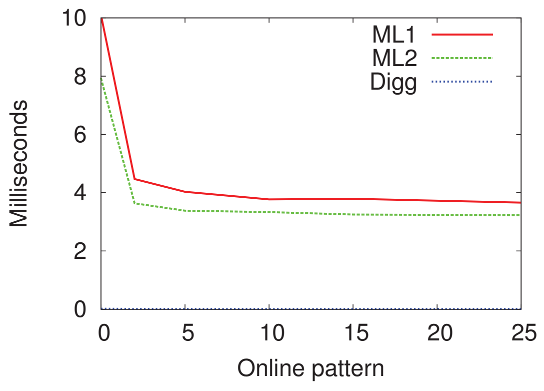


Figure 5.8: *DeRec* widget with varying online patterns.

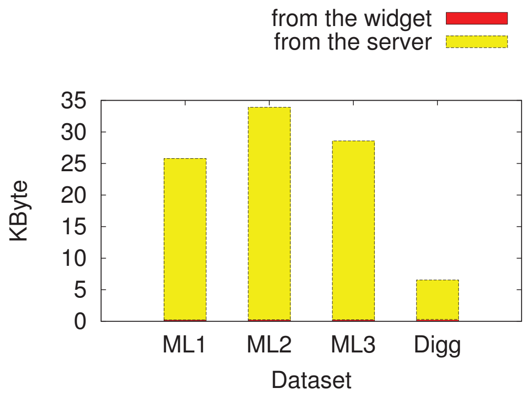
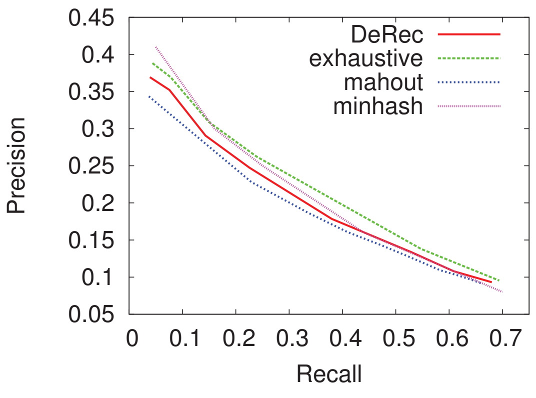


Figure 5.9: Communication overhead of *DeRec* (ML1).

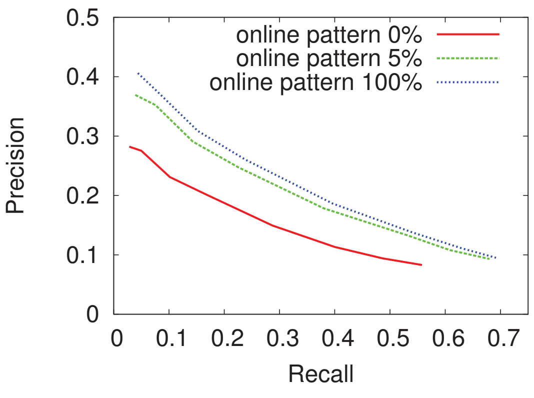
online patterns of 0 and other values on Figure 5.8.

5.6.7 Communication overhead

By delegating expensive computation tasks to clients, the *DeRec* server and widgets experience a communication overhead compared to a centralized architecture in charge of all the computation tasks. Figure 5.9 shows the average overhead generated by the server and the widgets according to the dataset, each time a user sends an online status to get recommended items. Results show that most of the overhead is generated by the server due to the sample message carrying user identifiers and their associated profiles. The clients only generate little overhead due to the notifications returned to the server when they have finished their tasks. However, this overhead is negligible compared to the average size of a Facebook page of 160.3 KBytes.



(a) According to approaches



(b) Impact of the online pattern

Figure 5.10: Recommendation analysis for ML1.

5.6.8 Recommendation quality

So far we have focused on computation time. We now evaluate the quality of recommendation to show that it is not impacted by *DeRec*'s hybrid architecture. First, we evaluate the precision-recall trade-off achieved by *DeRec* compared to the other centralized candidates. Figure 5.10a shows that the quality of the recommendation provided by *DeRec* is similar to *CRec*, Mahout, and the Minhash solution as presented in in [52]. As in *DeRec*, the k nearest neighbor selection is refined only when users are online, we evaluate the impact of the online pattern on recommendation. As shown on Figure 5.10b, the more the available users, the higher their chance to benefit from recommendation. However, only a small online pattern is needed to provide good recommendations: an online pattern of 5% gives about 95% of the maximum precision-recall. This is is mainly due to the fact that neighbor selection is almost as good when computed on a sample as on the entire dataset. Although not reported here for space reason, we observe that the size of the neighborhood does not impact significantly the quality of the recommendation.

5.7 Conclusions

This chapter explores a novel hybrid architecture for personalizing recommendations by combining the scalability and cost-effectiveness of massively distributed systems with the ease of management of centralized solutions. We convey the feasibility of the approach through *DeRec*, a generic user-based collaborative filtering system that can be adopted by various web applications. As shown by our exhaustive experiments on several datasets, *DeRec* indeed constitutes a viable and promising alternative to data-center based approaches by leveraging the hardware and computation power of client machines. Content providers can drastically reduce their cost by reducing the required resources dedicated to a personalized recommendation system. The improvement becomes even more noticeable as the workload increases.

As we pointed out, *DeRec* is generic in the sense that several of its aspects could be adapted to specific contexts while keeping the same hybrid architecture. The similarity metric, the actual recommendation algorithm and the way we sample the users can all be customized with minimal impact on the architecture. Also, while we focused on a k-nearest-neighbor solution, it would be interesting to see how matrix-factorization-based ones could be used in hybrid contexts. Finally, in the version presented in this chapter, *DeRec* potentially exposes a user's profile to other users. *DeRec* could be improved by hiding the association between a user and her profile and by having the server periodically shuffle this association or with mechanisms to avoid to require user's profile to compute the k-nearest neighbors as presented in Section 5.

Chapter 6

Conclusions and Perspectives

6.1 Summary

New web technologies have transformed the Internet into a user driven content generation platform where users are fully involved in the information processing from the creation to the evaluation and the dissemination. To guide a user to find relevant content among this huge and exponentially growing number of news items, recommendation systems that provide content personalization services have a central place in the Internet. However, providing personalization raises many challenges, especially to address the continuous stream of news items that need to be filtered and instantly delivered to users while preserving their privacy.

To address this challenge, we propose (*1*) WhatsUp, a fully distributed news recommender, (*2*) privacy-preserving mechanisms for distributed collaborative filtering, and (*3*) *DeRec*, a hybrid personalization system leveraging decentralization while using a centralized architecture.

Decentralizing news recommender

In Chapter 3, we first convey the feasibility of a distributed news personalization system, where every node dynamically collaborates with other nodes to achieve effective on-line personalized dissemination. We implemented such as system: WhatsUp. The personalized dissemination provided by WhatsUp is user-centric and collaborative at its very architectural level. Each user in the system dynamically builds and maintains her own implicit social network based on the opinions she expresses about the news items she receives (like-dislike). To achieve this, users leverage an asymmetric similarity metric reflecting long-standing and emerging (dis)interests to identify its implicit acquaintances in terms of interest. The push model of this metric (*i.e.* users choose the next hops to disseminate their news items, but have no control on who sends items to them), avoids fragmenting the interest-based topology into several disconnected parts and avoids node concentration around hubs, and speeds up the cold-start of users. WhatsUp disseminates news items through a novel heterogeneous gossip protocol that both biases the choice of its targets towards those with similar interests and amplifies dissemination based on the level of interest in every news item. These mechanisms of orientation and amplification adapt the propagation of each news item according to its dissemination path and its location in the system (*i.e.* two copies of the same item along two different paths act differently). Additionally, we show the benefits of these mechanisms to give items a chance to reach interested nodes across the entire network. Our exhaustive experiments show that WhatsUp, while relying only on partial knowledge, achieves a good trade-off between the accuracy and completeness of dissemination.

Privacy-preserving distributed collaborative filtering

While WhatsUp does not address privacy concerns, in chapter 4, we propose mechanisms to control the trade-off between privacy and recommendation quality in distributed recommender systems. Our mechanisms rely on two components: (*1*) an obfuscation technique applied to user profiles, and (*2*) a randomization-based dissemination algorithm ensuring differential privacy during the dissemination process. Each component applies to one of the core component of a decentralized user-based CF system: the user-clustering and the dissemination protocols. Our obfuscation protocol constructs filters to identify and exchange only the least sensitive parts of user profiles to form the neighbourhood of each user. By controlling the size of these filters, system designers can therefore tune the amount of sensitive information exchanged between users. Our randomization-based dissemination protocol, in turn, prevents any user from knowing with certainly the exact interest of the users who have sent a news item. Results show that our mechanism achieves similar results as a differentially private alternative in terms of recommendation quality while protecting the sensitive information of users more efficiently and being more resilient to censorship attacks.

Democratizing personalization

Fully decentralized approaches have been mainly exploited for non commercially-driven platforms, providing collaborative applications. Contrary to this, websites and content providers prefer a centralized controlled architecture to gather full knowledge and to monetize the data easily. In chapter 5, we explore *DeRec*, a hybrid personalization system leveraging decentralization while using a centralized architecture. *DeRec* democratizes the recommendation process by enabling content-providers to offer personalized services to their users at a minimal investment cost. *DeRec* provides the scalability of decentralization with the manageability of centralized solutions. *DeRec* can be easily integrated to existing solutions and consists of a lightweight back-end manager capable of transparently offloading CPU-intensive recommendation tasks to front-end user browsers. In addition, our solution is generic, content providers can customize *DeRec* to adapt it to their specific personalization needs. *DeRec* also incorporates an adaptation mechanism that can tune the computational tasks on both the server and the clients to cope with the capability of the client device and allows the server to operate effectively even during load peaks. Our extensive evaluation conveys its ability to drastically lower the operation costs of a recommendation system while preserving the quality of personalization when compared to a classical approach.

6.2 Ongoing works

While this work has provided some foundations to build privacy-aware distributed news recommender as well as to democratize the recommendation process, many challenges still to be addressed. Our WhatsUp and *DeRec* prototypes will certainly be key elements used to identify and overcome limitations. This section lists the ongoing works.

Temporal dimension

The dynamics of user participation, or churn, are an inherent property of social networks and P2P systems [91,134]. In a user-based dissemination context, the arrival and departure of users can have a strong impact on the way information is spread [37]. As this information is not publicly available in most social networks, we relied on digg dataset to compute and propose a model for describing the latency between the moment a user is exposed to a piece of news (*i.e.* receive a notification from a friend) and the moment at which this user "diggs" it. Regression analysis performed on the user latency of more than 30.000 users on several hundred of news between their submission and the moment they become popular (popular items in digg benefit from a better visibility which introduces a bias in their temporal activity) have shown that this latency follows the Gamma distribution with these parameters:

$$y = ysum * \Gamma(\frac{t}{div}, \alpha, \beta)$$

The parameter *ysum* is the number of diggs on the sample, $\alpha = 7.457e - 01$, $\beta = 2.848e - 03$ and $div = 1.018e + 00$.

Further studies using this model are currently being conducted to analyze the impact of churn in WhatsUp and to examine the temporal dimension of its dissemination.

Large scale experiments

Many social networks gather millions of users and items. A current limitation of WhatsUp is the lack of large scale experiments. This limitation is mainly due to both (*1*) the lack of datasets without dissemination bias introduced by the explicit nature of the topology of the underlying social network and (*2*) the difficulty for our simulator to handle millions of nodes.

Scaling up the experiments is a real challenge, we are currently adapting our simulator to handle more users and items. In addition, we analyzed users' interests in the survey dataset and generated a tree of conditional probabilities modeling their level of rating in one category according to their ratings in other categories. Then, we used this model to scale our dataset by generating new instances of user and news items while respecting the same global behavior as the original ones.

Identify communities

As discussed in Chapter 3, WhatsUp performs best when user communities are disjoint. While real datasets do not exhibit such communities, an interesting avenue of research would be to investigate solutions that somehow separate communities, potentially allowing nodes to be part of several ones in the form of virtual instances. This is particularly challenging when no explicit classification is available or desirable.

We conducted some experiments on the survey dataset to identify more precisely communities of interests. Specifically, we analyzed the similarity between users according to the popularity of items they liked (Figure 6.1) where the popularity of an item is measured as the proportion of users who liked it. Results clearly show that as an item gains more popularity, the higher the average similarity between users who liked them decreases. In other words, the more popular an item is, the more users from heterogeneous communities of interest like it. This suggests that too popular items should not be taken into consideration in user profiles during the clustering phase of Wup.

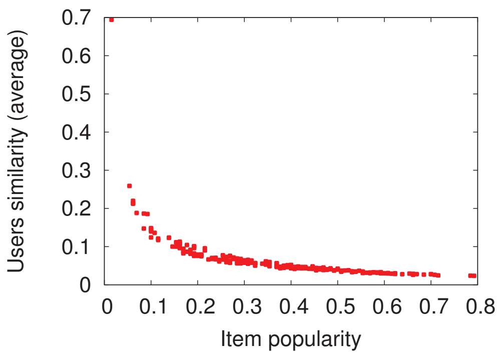
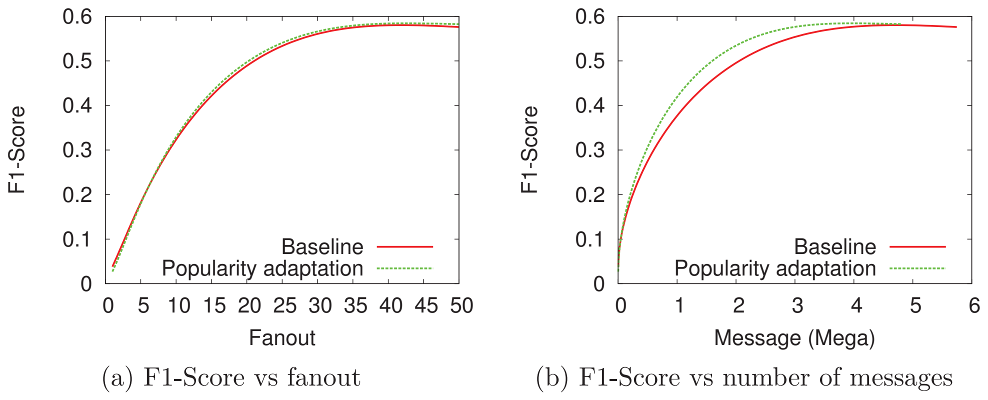


Figure 6.1: *Survey: similarity between users vs popularity of items*

From this observation we also conducted experiments to adapt the fanout for each news item according to its popularity. More specifically, at each cycle we computed the popularity of every item and, when a user likes an item, the value of the fanout is



(a) F1-Score vs fanout (b) F1-Score vs number of messages

Figure 6.2: *Survey: adaptation of the fanout according to the popularity of items*

inversely proportional to its popularity. For instance, an item with a popularity of 0.5 will have a fanout of $\frac{f_{\text{LIKE}}}{2}$, while the fanout will be close to 0 for a very popularity item. Figure 6.2a and Figure 6.2b depict the F1-Score according to the fanout and the number of messages, respectively. WhatsUp using cosine similarity represents the baseline for the comparison. Results show that leveraging item popularity to adapt the fanout tends to increase the F1-Score and drastically reduces the message cost.

Privacy by similarity challenges

In a distributed recommender system, users have to be able to compute their similarity with others to identify the k-nearest neighbors in term of interests. While we used obfuscation mechanisms in Chapter 4 to avoid revealing the real interests of users, privacy can be also achieved by using a similarity challenge without exchanging profiles. More precisely, in this model, user profiles are stored locally at every node in the form of a d-dimensional bit array similar to the compact profile presented in Section 4.8.1. The similarity challenge between two users consists of first doing a Diffie-Hellman key exchange to share secretly an integer between them. This integer is then secondly used by each user as a seed to initiate a random generator and to generate the same range of several random profiles. Finally, users compute and exchange a vector containing the similarity scores between their profile and all the random profiles. The size of the similarity challenge, noted ssc, is the number of random profiles used to form this vector. Users with the best average similarity score are used as neighbors in the implicit social network.

We conducted experiments with WhatsUp using both compact profiles to store the interests of users and similarity challenges using cosine similarity to maintain the implicit social network of Wup. Figure 6.3 depicts the F1-Score depending on the message cost for various sizes of similarity challenges (ssc). Results show that the higher the ssc, the more important the F1-Score. In addition, the solution using the

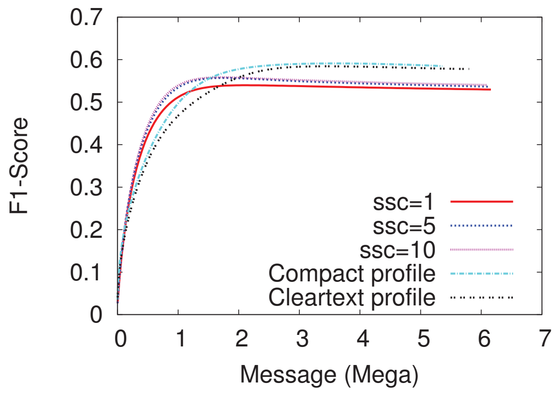


Figure 6.3: *Survey: F1-Score depending on message cost for various size of similarity challenge (ssc)*

similarity challenge converges faster than the solution using cleartext and compact profile, however, the F1-Score remains low.

This similarity challenge can be used for WhatsUp as described above or to protect *DeRec*. Indeed, in the current version of *DeRec*, users are exposed to the profile of other users. In order to avoid that, it would be interesting to explore the possibility of storing locally on each browser the interest profile of the corresponding user, where the centralized server takes care in performing similarity challenges without manipulating and storing user profiles.

Leveraging the content to speed up the cold start

Collaborative filtering is the prevalent approach to implement recommender systems. The reason behind the popularity of this method resides in its content-agnostic nature which does need item descriptions to uncover their relations, but take into account only similar users' past behavior to enable the recommendation. However, in collaborative filtering systems, the presence of new users and new items leads to the so-called cold-start issue.

WhatsUp is a purely user oriented collaborative filtering system and does not leverage the content (*e.g.* keyword extraction from news items [127]). It would be interesting to explore the benefits that can be extracted from the content in a concrete setting. We did experiments on the survey dataset to use the content of the associated news items to form the user profiles. More precisely, we used the basic term frequency-inverse document frequency technique (TF-IDF) [121] to attach a numerical statistic value that reflects the importance of words to each news item. The user profiles are then built using the most important words of each liked news item which drives the formation of the implicit social network of users.

Figure 6.4 shows the F1-Score depending on the message cost for WhatsUp using its purely user oriented collaborative filtering scheme (cleartext profile) and only a

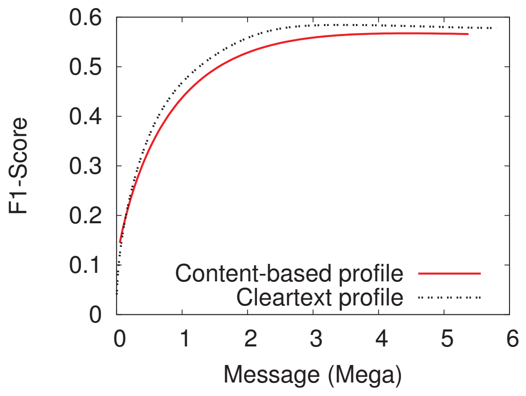


Figure 6.4: *Survey: F1-Score depending on message cost for both CF and content-based recommendation*

content-based strategy leveraging the TF-IDF technique. Results show that a purely content-based solution reduces slightly the F1-Score. However, as it is common to use different recommendation techniques together to alleviate the drawbacks of one with the advantages of the others [51, 103, 120], merging the content-based and the collaborative filtering strategies in WhatsUp could certainly speed up the cold start. In addition, leverage the content could be also useful to identify news items related to the same event and potentially merge them to provide only one news item.

An anonymous and distributed recommender

User privacy can also be achieved by anonymity [53, 54]. For these reasons, we have started to explore how to completely hide the association between a user and her ID (or IP address) through a proxy-based solution, inspired by Onion routing [59] and the anonymization method of Gossple [32]. A proxy, itself hidden by intermediary nodes, performs all exchanges on behalf of a user. However, the encryption and the multi-hop forwarding paths significantly amplify the cost in terms of consumed bandwidth. In our experiments, the use of one proxy and one intermediate node for each WhatsUp node leads to a 10-fold increase in the bandwidth requirements of both Wup and Beep.

6.3 Perspectives

Leveraging implicit and explicit acquaintances

Relying on explicitly declared friends, as in Digg, is known to limit the content that can be received [147] by substantially influencing decision making [88]. Consequently, the explicit nature of Digg introduces a bias in the profile of users since news is disseminated through the social structure. Basically, users are only exposed to the content forwarded by their friends, while other news may be of interest to them. However, sharing content with explicit friends is an important feature of most of existing social networks and is well appreciated by users. Leveraging an explicit and an implicit social network to provide the personalization would be an interesting avenue for WhatsUp.

Estimating the size of interest-based communities

Several works have been proposed for both estimating sizes of social networks via different sampling strategies and random walkers [73, 92, 149], and for counting the number of peers in a P2P system [105]. However, as far as we know, no work has proposed an estimator to evaluate the size of interest-based communities. The highly dynamic nature of news items makes this difficult to achieve, especially if a user can change her community very fast or can be part of different communities. In WhatsUp, the node's knowledge of the size of communities sharing similar interests by nodes would allow them to dynamically adapt the fanout to be as close as possible to the optimum value. For instance, the periodical sampling achieved by Wup could be leveraged to estimate this size.

Theoretical framework for WhatsUp

Understanding the spread of information through populations is of great importance. We do not understand, for instance, why some information captures the interest of communities while other information is largely ignored. Observation of social networks and user behaviors has attracted lots of interest these last years to analyze and model information diffusion [18–20, 42, 60]. In many cases, this turns out to be closely connected to the modeling of the spread of diseases and epidemics. To keep the mathematics tractable, these models tend to assume homogeneous populations: every node has the same strategy when gossiping. To the best of our knowledge, WhatsUp is the first gossip protocol to provide heterogeneity along multiple dimensions: amplification and orientation. It would be interesting to propose a theoretical framework for WhatsUp with multiclass gossip protocols mixing populations with different strategies.

Chapter 7

French summary

Introduction

L'évolution rapide du web a changé la façon dont l'information est créée, distribuée, évaluée et consommée. En effet, la révolution 2.0 du web a mis l'utilisateur au centre du web. Les utilisateurs représentent dorénavant la plus grande bande passante consommée, les ultimes décideurs de la réelle adoption des applications, et peut-être le plus important, le plus prolifique générateur de contenu.

Chaque événement, à n'importe quel endroit du globe, est instantanément commenté et débattu sur le web par les utilisateurs. Les conséquences sont drastiques. Même les révolutions sont initiées sur le web, et les nouvelles constitutions sont travaillées et débattues en temps réel.

Sans surprise, la tentation de s'informer directement par le web est plus forte que jamais et plusieurs d'entre nous y succombent. Une majorité de personnes dépense un temps considérable à consulter les news en navigant sur le web mais la quête n'est pas toujours efficace : le flot d'informations disponible est énorme et grandit exponentiellement.

Cette thèse prend place dans ce contexte et essaie de répondre à cette question : *Comment pouvons-nous filtrer le contenu non désiré tout en étant capable de découvrir de nouvelles informations pertinentes ?*

Cette question fait appel aux systèmes de recommandations pour la personnalisation des news. Tandis que les systèmes de recommandations ont été largement utilisés de plusieurs façons depuis leur introduction, le principal challenge dans le contexte de la personnalisation de news est de faire face au nombre croissant d'utilisateurs et de contenus ainsi que le caractère hautement dynamique de l'information.

De nos jours, deux principales techniques sont largement utilisées pour produire de la personnalisation aux utilisateurs : tirer parti des réseaux sociaux et exploiter l'analyse du comportement utilisateur.

Les réseaux sociaux tels que Twitter et Facebook ont connu une grande popularité et jouent un rôle majeur dans la dissémination d'informations tout autour du globe [13,15].
Dans la plupart des réseaux sociaux les utilisateurs déclarent explicitement des amis et partagent du contenu ou des images entre eux. De plus, la majorité des plate-formes sociales fournissent une interface pour faciliter le partage de contenu (créé, découvert sur des sites web externes ou découvert sur d'autres applications) avec les membres de son réseau. Ainsi, il est possible pour un utilisateur de recommander une news ou de l'information à ses amis.
Sous cette forme, les utilisateurs comptent sur leur réseau social et sa structure pour recevoir du contenu et des news personnalisées en fonction de leurs centres d'intérêts. Cependant, les informations importantes et les news les plus pertinentes proviennent souvent de personnes extérieures à son cercle direct d'amis. De plus, compter uniquement sur ses amis déclarés de façon explicite limite considérablement le contenu qu'un utilisateur peut recevoir [147] tout en introduisant potentiellement du spam (contenus non désirés). Par ailleurs, déclarer une personne "amie" un jour ne signifie pas que ses notifications et que les news qu'il postera quelques mois plus tard seront pertinentes.

D'un autre côté, une partie importante des sites web et des applications modernes ne compte pas uniquement sur les réseaux sociaux, mais collecte et tire parti de toutes les informations des utilisateurs. Les données collectées peuvent ainsi être exploitées à différents buts.
Par exemple, Digg laisse les utilisateurs soumettre du contenu ainsi qu'exprimer leurs opinions sur les articles disponibles afin d'identifier l'information la plus populaire. On parle de "crowdsourcing" quand les actions et l'information proposées aux utilisateurs sont issues d'activité collaborative.
De telles informations peuvent aussi être utilisées pour construire des modèles de comportement utilisateur. Par exemple, Facebook et Google tirent parti de boutons attachés aux sites web offrant une interface avec leur réseau social respectif pour suivre la navigation des utilisateurs sur le web. Les informations collectées leur permettent d'améliorer la connaissance des utilisateurs à travers leurs goûts et leurs habitudes dans le but d'améliorer l'efficacité de leurs moteurs de recherche et leurs systèmes de personnalisation.
Les informations sur les utilisateurs peuvent aussi être échangées et monayées auprès de différentes compagnies pour proposer des publicités personnalisées [1]. Plus un système a de connaissances sur les intérêts de ses utilisateurs, plus il est capable de lui fournir des personnalisations pertinentes. Cependant, ce monitoring intrusif a tendance à effrayer les utilisateurs et peut les dissuader d'exprimer librement leurs opinions et leurs préférences par crainte d'être épié, ce qui rend, par conséquence, le système collaboratif de recommandations totalement inefficace et inutile.

Tandis que la personnalisation améliore grandement l'expérience des utilisateurs, elle soulève de multiples questions et challenges d'un point de vue de la vie privée. Par exemple, un système de personnalisation sur une plate-forme sociale peut révéler des informations susceptibles d'être embarrassantes directement aux amis, membres de la famille ou collègues d'un utilisateur.

De plus, les utilisateurs perdent le contrôle de leurs données privées et de la manière dont elles sont échangées et exploitées. Facebook, par exemple, a changé les règles d'utilisation de son réseau social en 2007 dans le but de s'octroyer la propriété de l'ensemble du contenu utilisateur de sa plate-forme. Ceci afin de pouvoir l'exploiter commercialement. Plus récemment, fin 2012, Instagram a changé ses termes d'utilisation afin qu'il puisse vendre les photos de ses utilisateurs sans leur accord. Les dérives de propriété et les utilisations commerciales abusives et non éthiques des données augmentent de plus en plus l'inquiétude des utilisateurs.

D'autre part, les systèmes de recommandations sont en grande partie centralisés. Cette architecture a l'avantage de permettre aux fournisseurs de bénéficier d'une connaissance globale sur les utilisateurs et le contenu. Cependant, ce modèle fait appel à des approches nécessitant de grandes capacités de calculs dont seules les grandes compagnies peuvent disposer, à cause du coût du passage à l'échelle nécessitant des solutions centralisées sur le "cloud".

Certains acteurs relatent aussi leurs problèmes pour faire face au caractère hautement dynamique du contenu [52]. En effet, les calculs pour déterminer le voisinage des utilisateurs en terme de centre d'intérêts se font de manière différée. Les mises à jour ne sont donc pas en temps réel.

De plus, une architecture centralisée comporte des vulnérabilités. En effet, un système accueilli sur un groupe de serveurs peut être confronté à des attaques ou rencontrer des problèmes de passage à l'échelle [14]. Par ailleurs, les événements récents en Tunisie, Égypte ou Syrie ont souligné l'importante limite des applications centralisées face au filtrage et à la censure. Les censeurs peuvent facilement filtrer un réseau social ou un système de partage de vidéos pour bloquer le débat politique ou prévenir la dissémination d'informations, si celui-ci est centralisé.

D'un autre côté, les réseaux pair-à-pair ont largement émergé, fournissant des fondations stables pour des systèmes à grande échelle sur lesquels il est possible de construire des applications distribuées.

Premièrement, les systèmes pair-à-pair peuvent fournir une alternative bien moins onéreuse que de grands systèmes basés sur des data-centers énergivores. En effet, le modèle de passage à l'échelle des data-centers provient, en règle générale, du nombre des ressources qu'une compagnie dédie au système.

Deuxièmement, grâce à leur structure distribuée, les solutions pair-à-pair peuvent devenir une alternative intéressante contre les compagnies intrusives pour fournir une plate-forme collaborative et un système de dissémination d'informations respectant la vie privée des utilisateurs.

Pour toutes ces raisons, nous avons exploité dans cette thèse les architectures pair-à-pair pour implémenter des systèmes de recommandations.

Challenges

Le principal challenge de cette thèse est de proposer un système de recommandation distribué efficace pour la personnalisation de news, tout en respectant la vie privée des utilisateurs.

Contributions de cette thèse

La motivation de ces travaux est d'explorer la faisabilité d'un système de personnalisation de news, centré sur l'utilisateur et collaboratif jusque dans son architecture. Le but est de regrouper les utilisateurs autour d'intérêts communs et de fournir une dissémination d'informations personnalisée et efficace. Pour cela, nous avons conçu et développé une solution où chaque noeud conserve et protège le profil d'intérêts de l'utilisateur tout en collaborant dynamiquement avec les autres noeuds.

Dans ce contexte, les approches pair-à-pair sont attractives car elles passent à l'échelle naturellement et évitent qu'une entité centrale contrôle tous les profils des utilisateurs et potentiellement les exploite à but commercial. Cependant, l'absence de connaissance globale fait appel à des schémas de filtrage collaboratif qui doivent palier les informations partielles et dynamiques des utilisateurs. De plus, concevoir un schéma de filtrage efficace, respectant la vie privée, simple et assez léger pour être facilement déployable sur des machines personnelles sans aide d'un serveur central reste un challenge.

WhatsUp : un système de recommandation de news distribué

La première contribution de cette thèse est un système de recommandations de news distribué, nommé WhatsUp. WhatsUp maintient dynamiquement un réseau social implicit basé sur les opinions que les utilisateurs expriment à propos des news qu'ils reçoivent (j'aime/je n'aime pas). Pour cela, chaque noeud échange périodiquement son profil d'intérêt (liste des news reçues ainsi que l'opinion de l'utilisateur associé) avec d'autre noeud du réseau afin d'identifier son voisinage en terme de centre d'intérêt. La mesure de similarité considérée prend en compte les intérêts latents et émergents des utilisateurs, et évite la formation d'une structure non connexe.

Les news sont disséminées au travers d'un nouveau protocole épidémique hétérogène qui (*1*) biaise l'orientation des cibles à l'égard des personnes de son voisinage et introduit de la sérendipité (habilité à découvrir du contenu inattendu et intéressant) et (*2*) amplifie ou réduit la dissémination de chaque news en fonction du niveau d'intérêt qu'elle suscite. Ces mécanismes d'orientation et d'amplification permettent notamment d'améliorer les chances de chaque news à atteindre des noeuds intéressés à travers le réseau.

Par souci de clarté, dans cette première partie, nous n'abordons pas les aspects de protection de vie privée mais nous nous concentrons à montrer que la recommandation de news peut être efficacement effectuée par un système distribué pour filtrer de manière collaborative et disséminer des news de manière personnalisée.

Système de filtrage collaboratif distribué respectant la vie privée

Bien que la nature décentralisée des systèmes pair-à-pair supprime le problème des compagnies intrusives ayant un accès total aux données des utilisateurs, les fuites d'informations à caractère privé peuvent venir d'autres utilisateurs si le schéma de filtrage nécessite la manipulation de données personnelles.

La seconde contribution de cette thèse propose (*1*) un mécanisme d'offuscation cachant le profil exact des utilisateurs sans trop dégrader son utilité (pour l'identification des utilisateurs ayant des centres d'intérêt similaires), et (*2*) propose un processus aléatoire de dissémination.

Plus précisément, notre protocole d'offuscation construit un filtre qui identifie les informations les moins sensibles du profil d'intérêt de l'utilisateur. En contrôlant la taille de ce filtre, le designer du système peut faire varier la quantité d'information échangée entre les utilisateurs pour former leur voisinage. Notre dissémination à caractère aléatoire, de son côté, évite qu'un utilisateur puisse savoir avec certitude l'opinion exacte des utilisateurs qui lui envoient des news.

DeRec : Démocratisation des systèmes de personnalisation

Un système de recommandation de news totalement distribué comporte plusieurs avantages du point de vue du passage à l'échelle et de la résilience aux pannes. Il permet aussi aux utilisateurs de librement l'exploiter sans aucune forme de contraintes liées à la rémunération du système, tel que la publicité par exemple. Cependant, les sites web commerciaux et les éditeurs de contenu sur le web n'ont pas encore trouvé de modèles économiques adaptés à cette architecture distribuée et préfèrent les systèmes centralisés pour facilement exploiter l'ensemble des données.

La dernière contribution de cette thèse explore un nouveau modèle tirant parti des avantages des systèmes distribués tout en conservant une architecture centralisée. *DeRec* démocratise les systèmes de recommandations en offrant aux fournisseurs de contenu un système de personnalisation pour leurs utilisateurs, et ce à faible coût.

DeRec fait appel à une architecture hybride composée d'un gestionnaire de tâches léger capable d'exporter la partie la plus importante du processus de recommandation sur le navigateur des utilisateurs. *DeRec* intègre également un mécanisme qui permet d'adapter les tâches effectuées par le serveur et par les clients en fonction de la charge du serveur et de la capacité de calcul du matériel de l'utilisateur. Notre solution est générique et peut facilement être adaptée à des systèmes existants.

Les contributions de cette thèse sont résumées sur la figure suivante 7.1.

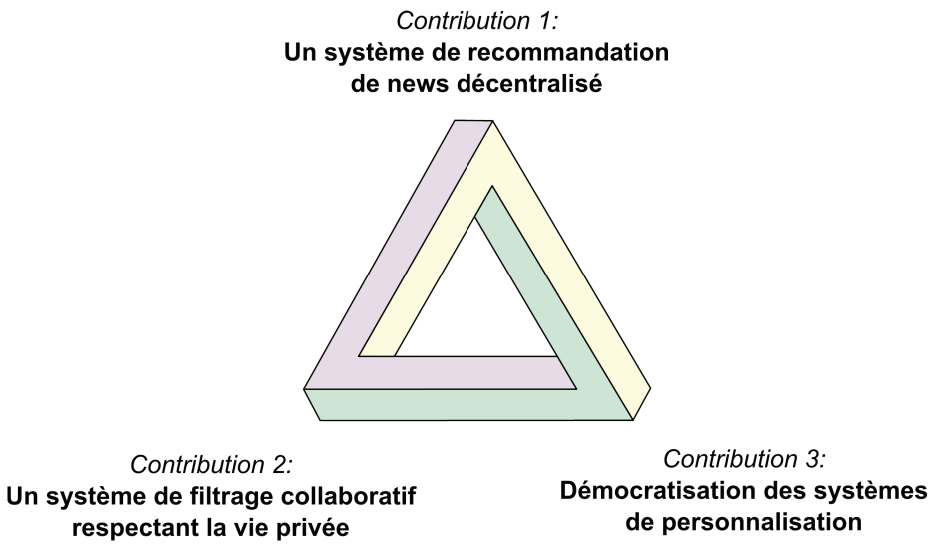

Contribution 1:
Un système de recommandation de news décentralisé
Contribution 2:
Un système de filtrage collaboratif respectant la vie privée
Contribution 3:
Démocratisation des systèmes de personnalisation

FIGURE 7.1 – *Contribution de cette thèse : 1) un système de recommandation de news distribué, 2) des mécanismes de protection de vie privée pour les systèmes de filtrages collaboratifs distribués, et 3) un modèle hybride pour démocratiser les systèmes de personnalisation.*

Autres contributions

Durant ma thèse, j'ai aussi analysé la disponibilité des utilisateurs dans les réseaux sociaux et proposé un système de classification de l'orientation politique des utilisateurs de Twitter durant les élections. Ces contributions ne sont pas décrites dans ce document mais sont résummées ci-dessous.

L'impact de la disponibilité des utilisateurs dans les réseaux sociaux. [37]

La disponibilité des utilisateurs peut avoir un impact important sur des applications en ligne. Basés sur des traces collectées à partir de MySpace, nous avons montré que la disponibilité des utilisateurs tend à être corrélée à celles de leurs amis.

Nous avons aussi mis en avant le rôle prépondérant de la disponibilité des utilisateurs dans la diffusion d'informations. En effet, les utilisateurs ayant une position centrale dans la structure du réseau social ont une grosse influence sur la dissémination. En fonction de la disponibilité des utilisateurs, leur position et donc leur importance dans le processus de dissémination peut évoluer de manière importante.

Les caractéristiques des partis politiques et la polarisation des utilisateurs sous Twitter [38–40]

Dans la politique moderne, les partis et les candidats doivent avoir une présence sur le web ainsi que sur les réseaux sociaux. Dans ce contexte, nous avons étudié l'avantage de l'analyse des messages Twitter pour identifier les caractéristiques des partis politiques et l'orientation politique des utilisateurs. Basés sur le flot de messages Twitter durant la période des élections anglaises de 2010, nous avons examiné les caractéristiques des trois principaux partis politiques disputant ces élections et mis en avant leurs principales différences.

De ces observations, nous avons développé une méthode de classification incrémentale pouvant être utilisée en temps réel qui utilise uniquement l'activité des Tweets. Les expérimentations montrent que la méthode proposée fournit une bonne précision et surpasse les autres méthodes de classification dans ce domaine qui nécessitent soit un coût important pour l'entraînement du système de classification, soit la connaissance globale de la structure du réseau social.

Travaux en cours et perspectives

Les travaux de cette thèse produisent un base solide pour la construction et la démocratisation des systèmes de recommandations respectant la vie privée des utilisateurs, cependant plusieurs challenges doivent encore être surmontés. Le prototype de WhatsUp et de *DeRec* sont des outils très utiles pour identifier et surmonter ces limites. Voici une liste de travaux en cours ainsi que quelques perspectives :

- Dimension temporelle : la disponibilité des utilisateurs a un impact important sur la dissémination de l'information dans un système [37]. Basés sur un modèle construit à partir de traces venant de Digg, nous conduisons des expérimentations pour analyser l'impact de la disponibilité des utilisateurs sur la dissémination issue de WhatsUp.

- Expérimentations large échelle : les réseaux sociaux peuvent rassembler plusieurs millions d'utilisateurs. WhatsUp manque d'expérimentations large échelle principalement dû à l'absence de jeux de données non biaisés par la structure du réseau social sous-jacent ainsi qu'à la limite de notre simulateur. Des expérimentations sont en cours pour adapter notre simulateur et construire un modèle pour le profil d'intérêt des utilisateurs ainsi que la popularité des news à partir du jeu de données de notre questionnaire.

- Montré au chapitre 3, WhatsUp se comporte mieux quand les communautés d'intérêts sont bien distinctes. Une perspective intéressante serait d'explorer des solutions permettant de mieux dissocier les communautés d'utilisateurs ainsi que de donner la possibilité aux utilisateurs de faire parti de plusieurs communautés en fonction de leurs intérêts.

- Challenge de similarité : afin d'améliorer la vie privée des utilisateurs dans les systèmes de recommandations, nous travaillons sur une solution effectuant des challenges de similarité sur les profils d'intérêts au lieu de les échanger entre utilisateurs.

- Démarrage à froid : le démarrage à froid est un problème inhérent aux systèmes de recommandation quand un objet ou un utilisateur détient un profil vide. Les premières expérimentations, tirant parti du contenu des news, fournissent des résultats intéressants. Des travaux sont en cours pour intégrer ces mécanismes au problème du démarrage à froid dans WhatsUp.

- Système de recommandations anonyme : la protection de la vie privée peut aussi venir de l'anonymat. Une solution de système de recommandation dans laquelle les utilisateurs collaborent entre eux de manière anonyme est en phase de test.

- Amis implicits et explicits : Tirer parti d'un système comportant uniquement des amis explicits est connu pour limiter le contenu qui peut être reçu [147] et influence les décisions des utilisateurs [88]. Cependant, cette fonctionnalité est très appréciée par les utilisateurs. Mixer amis explicits et amis implicits peut être une piste intéressante pour WhatsUp.

- Estimation de la taille des communautés : plusieurs travaux ont été proposé pour l'estimation du nombre de noeuds dans un réseau pair-à-pair [105] et du nombre d'utilisateurs dans un réseau social [73, 92, 149]. Un estimateur de la taille des communautés dans WhatsUp permettrait de mieux adapter l'amplification de la dissémination au sein des communautés.

- Cadre théorique de WhatsUp : une compréhension fine de la dissémination d'informations dans une population serait très utile et a déjà suscité beaucoup de travaux [18–20, 42, 60]. En effet, de nombreux phénomènes restent encore inexpliqués. Souvent, les modèles de dissémination d'informations se rapprochent des modèles épidémiques. Pour limiter la complexité, ces modèles assument le plus souvent des populations et des comportements homogènes contrairement au protocole de dissémination de WhatsUp. Une perspective intéressante serait de proposer un cadre théorique pour WhatsUp avec un protocole épidémique mixant des populations comportant des stratégies différentes.

Bibliographie

[1] Collusion : Discover who's tracking you online. `http://www.mozilla.org/en-US/collusion/`.

[2] Digg. `http://digg.com`.

[3] Gnutella. `http://gnutella.wego.com`.

[4] Hadoop. `http://hadoop.apache.org/`.

[5] Jackson. `http://jackson.codehaus.org/`.

[6] Jetty. `http://www.eclipse.org/jetty`.

[7] Kazaa. `http://www.kazza.com`.

[8] Likelike. `http://code.google.com/p/likelike`.

[9] Mahout. `http://mahout.apache.org`.

[10] Movielens. `http://www.grouplens.org/node/73`.

[11] Recsys. `http://recsys.acm.org/`.

[12] Skype. `http://www.skype.com`.

[13] Twitter helps in haiti quake coverage, aid. `http://blogs.wsj.com/digits/2010/01/14/twitter-helps-in-haiti-quake-coverage-aid/`.

[14] Twitter is over capacity. `https://twitter.com/503`.

[15] U.s. state department speaks to twitter over iran. `http://www.reuters.com/article/2009/06/16/us-iran-election-twitter-usa-idUSWBT01137420090616`.

[16] I. Aekaterinids and P. Triantafillou. Pyracanthus : A scalable solution for dht-independent content-based publish/subscribe data networks. *Information Systems*, 2010.

[17] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS*, 2001.

[18] D. Agrawal, B. Bamieh, C. Budak, A. El Abbadi, A. J. Flanagin, and S. Patterson. Data-driven modeling and analysis of online social networks. In *WAIM*, 2011.

[19] D. Agrawal, C. Budak, and A. El Abbadi. Information diffusion in social networks : observing and affecting what society cares about. In *CIKM*, 2011.

[20] D. Agrawal, C. Budak, and A. El Abbadi. Structural trend analysis for online social networks. *PVLDB*, 2011.

[21] M. Agrawal, M. Karimzadehgan, and C. Zhai. An online news recommender system for social networks. In *SIGIR-SSM*, 2009.

[22] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[23] W. Ahmad and A. Khokhar. An architecture for privacy preserving collaborative filtering on web portals. In *IAS*, 2007.

[24] M. Alaggan, S. Gambs, and A-M. Kermarrec. BLIP : Non-interactive Differentially-Private Similarity Computation on Bloom Filters. In *SSS*, 2012.

[25] K. Ali and W. van Stam. Tivo : making show recommendations using a distributed collaborative filtering architecture. In *KDD*, 2004.

[26] E. Ameur, G. Brassard, J. M. Fernandez, and F. S. Mani Onana. Alambic : a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Secur.*, 2008.

[27] X. Bai. Personalized top-k processing : from centralized to decentralized systems. In *Ph.D. thesis*, 2010.

[28] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *EDBT*, 2010.

[29] X. Bai, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Collaborative personalized top-k processing. *ACM Trans. Database Syst.*, 2011.

[30] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera : topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.

[31] E. Banos, I. Katakis, N. Bassiliades, G. Tsoumakas, and I. P. Vlahavas. Personews : a personalized news reader enhanced by machine learning and semantic filtering. In *ODBASE*, 2006.

[32] M. Bertier, D. Frey, R. Guerraoui, A.M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware*, 2010.

[33] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *IPTPS*, 2003.

[34] K. Bharat, T. Kamba, and M. Albers. Personalized, interactive news on the web. *Multimedia Systems*, 1998.

[35] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. Technical report, 1999.

[36] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[37] A. Boutet, A.-M. Kermarrec, E. Le Merrer, and A. Van Kempen. On the impact of users availability in osns. In *SNS*, 2012.

[38] A. Boutet, H. Kim, and E. Yoneki. Member classification and party characteristics in twitter during uk election. In *DYNAM*, 2011.

[39] A. Boutet, H. Kim, and E. Yoneki. What's in your tweets ? i know who you supported in the uk 2010 general election. In *ICWSM*, 2012.

[40] A. Boutet, E. Yoneki, and K. Hyoungshick. What's in Twitter : I Know What Parties are Popular and Who You are Supporting Now ! In *ASONAM*, 2012.

[41] M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM ICDM*, 2003.

[42] C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *WWW*, 2011.

[43] R Burke. Hybrid recommender systems : Survey and experiments. *User Modeling and User-Adapted Interaction*, 2002.

[44] J. Canny. Collaborative filtering with privacy. In *SP*, 2002.

[45] J. Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR*, 2002.

[46] J. Carretero, F. Isaila, A.-M. Kermarrec, F. Taïani, and J. Tirado. Geology : Modular Georecommendation in Gossip-Based Social Networks. In *ICDCS*, 2012.

[47] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC*, 2000.

[48] M. Castro, P. Druschel, A. M. Kermarrec, and A. I.T. Rowstron. Scribe : a large-scale and decentralized application-level multicast infrastructure. *IEEE J.Sel. A. Commun.*, 2006.

[49] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *EuroSys*, 2012.

[50] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast : a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.

[51] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper. In *SIGIR*, 1999.

[52] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization : scalable online collaborative filtering. In *WWW*, 2007.

[53] S. Das, O. Egecioglu, and A. El Abbadi. Anonymizing weighted social network graphs. In *ICDE*, 2010.

[54] S. Das, O. Egecioglu, and A. El Abbadi. Anónimos : An lp-based approach for anonymizing weighted social network graphs. *IEEE Trans. Knowl. Data Eng.*, 2012.

[55] A. Datta and R. Sharma. Godisco : selective gossip based dissemination of information in social community based overlays. In *ICDCN*, 2011.

[56] J. Dean and S. Ghemawat. Mapreduce : simplified data processing on large clusters. *Commun. ACM*, 2008.

[57] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, 1987.

[58] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *RecSys*, 2012.

[59] R. Dingledine, N. Mathewson, and P. Syverson. Tor : the second-generation onion router. In *USENIX Security Symposium*, 2004.

[60] M. Draief and L. Massoulié. *Epidemics and rumours in complex networks*. Cambridge University Press, 2009.

[61] C. Dwork. Differential privacy : a survey of results. In *TAMC*, 2008.

[62] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*. 2006.

[63] M. D. Ekstrand, J.T. Riedl, and J.A. Konstan. *Collaborative Filtering Recommender Systems*. Now Publishers, 2011.

[64] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *KDD*, 2011.

[65] P. Erdos and R. Renyi. On the evolution of random graphs. In *Publication of the Mathematical institute of the Hungaruan academy of sciences*, 1960.

[66] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *Computing Surveys*, 2003.

[67] P. T. Eugster, R. Guerraoui, S.B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *TOCS*, 2003.

[68] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulieacute ;. Epidemic information dissemination in distributed systems. *Computer*, 2004.

[69] D. Frey, R. Guerraoui, A.M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma. Heterogeneous gossip. In *Middleware*, 2009.

[70] Raman G. and Michael J.C. Extending map-reduce for efficient predicate-based sampling. In *ICDE*, 2012.

[71] R. Gemulla, E. Nijkamp, P.J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, 2011.

[72] G. Giuffrida and C. Zarba. A recommendation algorithm for personalized online news based on collective intelligence and content. In *ICAART*, 2011.

[73] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook : a case study of unbiased sampling of osns. In *INFOCOM*, 2010.

[74] I. Goiri, K. Le, T.D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Green-hadoop : leveraging green energy in data-processing frameworks. In *EuroSys*, 2012.

[75] GossipLib. `http://gossiplib.gforge.inria.fr`.

[76] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *SEC*, 2011.

[77] P. Han, B. Xie, F. Yang, and R. Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 2004.

[78] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 2004.

[79] T. Hofmann. Latent semantic models for collaborative filtering. *ACM TOIS*, 2004.

[80] Z. Huang, W. Chung, T.-H. Ong, and H. Chen. A graph-based recommender system for digital library. In *JCDL*, 2002.

[81] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *SIGMOD*, 2005.

[82] S. Isaacman, S. Ioannidis, A. Chaintreau, and M. Martonosi. Distributed rating prediction in user generated content streams. In *RecSys*, 2011.

[83] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving p2p data sharing with oneswarm. *SIGCOMM Computer Communication Review*, 2010.

[84] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *TOCS*, 2005.

[85] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. *LNCS*, 2005.

[86] M. Jelasity, A. Montresor, and O. Babaoglu. T-man : Gossip-based fast overlay topology construction. *Comput. Netw.*, 2009.

[87] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.v. Steen. Gossip-based peer sampling. *TOCS*, 2007.

[88] T. Kameda, Y. Ohtsubo, and M. Takezawa. Centrality in sociocognitive networks and social influence : an illustration in a group decision-making context. *Journal of Personality and Social Psychology*, 1997.

[89] P. Kanerva, J. Kristoferson, and A. Holst. Random indexing of text samples for latent semantic analysis. In *CCSS*, 2000.

[90] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *ICDM*, 2003.

[91] M. Karnstedt, T. Hennessy, J. Chan, and C. Hayes. Churn in social networks : A discussion boards case study. In *SOCIALCOM*, 2010.

[92] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. In *WWW*, 2011.

[93] A.-M. Kermarrec. Toward a personalized internet : a case for a full decentralization. In *Philosophical transactions of the royal society (to be appear)*, 2013.

[94] A.-M. Kermarrec, V. Leroy, A. Moin, and C. Thraves. Application of random walks to decentralized recommender systems. In *OPODIS*, 2010.

[95] J. A. Konstan and J. T. Riedl. Recommender systems : from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 2012.

[96] Y. Koren. Factorization meets the neighborhood : a multifaceted collaborative filtering model. In *SIGKDD*, 2008.

[97] Yehuda Koren. Collaborative filtering with temporal dynamics. In *SIGKDD*, 2009.

[98] K. Lang. Newsweeder : learning to filter netnews. In *ICML*, 1995.

[99] H.W. Lauw, J.C. Shafer, R. Agrawal, and A. Ntoulas. Homophily in the digital world : A livejournal case study. *IEEE Internet Computing*, 2010.

[100] Z. C. Lawrence and K. Takeo. Maximum entropy for collaborative filtering. In *UIA*, 2004.

[101] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM*, 2005.

[102] G. Linden, B. Smith, and J. York. Amazon.com recommendations : item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.

[103] Prem M., Raymond J. M., and Ramadass N. Content-boosted collaborative filtering for improved recommendations. In *AAAI*, 2002.

[104] A. Machanavajjhala, A. Korolova, and A. D. Sarma. Personalized social recommendations : accurate or private. *VLDB*, 2011.

[105] L. Massoulié, E. Le Merrer, A-M. Kermarrec, and A. G. Peer counting and sampling in overlay networks : random walk methods. In *PODC*, 2006.

[106] M. Matos, A. Nunes, R. Oliveira, and J. Pereira. Stan : exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *IPTPS*, 2010.

[107] S.M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough : how accuracy metrics have hurt recommender systems. In *CHI EA*, 2006.

[108] D. Meisner, C.M. Sadler, L.A. Barroso, W.D. Weber, and T.F. Wenisch. Power management of online data-intensive services. *SIGARCH Comput. Archit. News*, 2011.

[109] B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens : toward a personal recommender system. *TOIS*, 2004.

[110] L. Ming, Y. Fan, K. Minkyong, C. Han, and L. Hui. In *IPDPS*, 2011.

[111] A. Montresor, M. Jelasity, and O. Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *DSN*, 2004.

[112] A. Narayanan and V. Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, 2006.

[113] M.E.J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 2004.

[114] Jia P. and Dinesh M. Bi-level locality sensitive hashing for k-nearest neighbor computation. In *ICDE*, 2012.

[115] J. A. Patel, E. Rivière, I. Gupta, and A.-M. Kermarrec. Rappel : Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Comput. Netw.*, 2009.

[116] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *SIGKDD*, 2007.

[117] P.R. Pietzuch and J.M. Bacon. Hermes : a distributed event-based middleware architecture. In *ICDCS Workshops*, 2002.

[118] H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *ICDM*, 2003.

[119] H. Polat and W. Du. Svd-based collaborative filtering with privacy. In *SAC*, 2005.

[120] A. Popescul, Lyle H. Ungar, David M. P., and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *UAI*, 2001.

[121] Stephen R. Understanding inverse document frequency : On theoretical arguments for idf. *Journal of Documentation*, 2004.

[122] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi. Vitis : A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *IPDPS*, 2011.

[123] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona : a high performance publish-subscribe system for the world wide web. In *NSDI*, 2006.

[124] S. Ranganathan, A. D. George, R. W. Todd, and M. C. Chidester. Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters. *Cluster Computing*, 2001.

[125] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 2001.

[126] D. Rosaci, G.M.L. Sarné, and S. Garruzzo. Muaddib : A distributed recommender system supporting device adaptivity. *ACM TOIS*, 2009.

[127] M. Rossignol and P. Sébillot. Combining statistical data analysis techniques to extract topical keyword classes from corpora. *Intelligent Data Analysis, IOS Press*, 2005.

[128] A. I. T. Rowstron and P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[129] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *Trans. Netw.*, 2004.

[130] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *SIGOPS*, 2004.

[131] SNAP. stanford network analysis platform `http://snap.stanford.edu`.

[132] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures : a large-scale study in the orkut social network. In *KDD*, 2005.

[133] R. E. Strom, G. Banavar, T. D. Chandra, Kaplan M. A., K. Miller, B. Mukherjee, D. C. Sturman, and M. Ward. Gryphon : An information flow based approach to message brokering. *CoRR*, 1998.

[134] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *IMC*, 2006.

[135] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009.

[136] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining.* Addison Wesley, 2005.

[137] E. Toch, Y. Wang, and L. F. Cranor. Personalization and privacy : a survey of privacy risks and remedies in personalization-based systems. *User Modeling and User-Adapted Interaction*, 2012.

[138] P. Triantafillou and I. Aekaterinidis. Peer-to-peer publish-subscribe systems. *Encyclopedia of Database Systems*, 2009.

[139] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI*, 2002.

[140] C. J. van Rijsbergen. *Information retrieval.* Butterworth, 1979.

[141] S. Verma and W.T. Ooi. Controlling gossip protocol infection pattern using adaptive fanout. In *ICDCS*, 2005.

[142] S. Voulgaris, D. Gavidia, and M. v. Steen. Cyclon : inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 2005.

[143] S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par*, 2005.

[144] Bernard W. and Saikat G. Quasar : A Probabilistic Publish-Subscribe System for Social Networks. In *IPTPS*, 2008.

[145] M. Wan, A. Jönsson, C. Wang, L. Li, and Y. Yang. A random indexing approach for web user clustering and web prefetching. In *PAKDD*, 2012.

[146] Stanley L. Warner. Randomized response : a survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309) :63–69, March, 1965.

[147] F. Wu, B.A. Huberman, L.A. Adamic, and J.R. Tyler. Information flow in social groups. *Physica A : Statistical and Theoretical Physics*, 2004.

[148] S.H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha. Like like alike - joint friendship and interest propagation in social networks. In *WWW*, 2011.

[149] S. Ye and F. Wu. Estimating the size of online social networks. In *SOCIALCOM*, 2010.

[150] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse. Tribler : P2p media search and sharing. In *MM*, 2011.

[151] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry : An infrastructure for fault-tolerant wide-area location and. Technical report, 2001.

[152] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux : an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, 2001.

[153] C. Ziegler, S.M. McNee, J.A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.

Résumé

L'évolution rapide du web a changé la façon dont l'information est créée, distribuée, évaluée et consommée. L'utilisateur est dorénavant mis au centre du web en devenant le générateur de contenu le plus prolifique. Pour évoluer dans le flot d'informations, les utilisateurs ont besoin de filtrer le contenu en fonction de leurs centres d'intérêts. Pour bénéficier de contenus personnalisés, les utilisateurs font appel aux réseaux sociaux ou aux systèmes de recommandations exploitant leurs informations privées. Cependant, ces systèmes posent des problèmes de passage à l'échelle, ne prennent pas en compte la nature dynamique de l'information et soulèvent de multiples questions d'un point de vue de la vie privée. Dans cette thèse, nous exploitons les architectures pair-à-pair pour implémenter des systèmes de recommandations pour la dissémination personnalisée des news. Une approche pair-à-pair permet un passage à l'échelle naturel et évite qu'une entité centrale contrôle tous les profils des utilisateurs. Cependant, l'absence de connaissance globale fait appel à des schémas de filtrage collaboratif qui doivent palier les informations partielles et dynamiques des utilisateurs. De plus, ce schéma de filtrage doit pouvoir respecter la vie privée des utilisateurs. La première contribution de cette thèse démontre la faisabilité d'un système de recommandation de news totalement distribué. Le système proposé maintient dynamiquement un réseau social implicit pour chaque utilisateur basé sur les opinions qu'il exprime à propos des news reçues. Les news sont disséminées au travers d'un protocole épidémique hétérogène qui (1) biaise l'orientation des cibles et (2) amplifie la dissémination de chaque news en fonction du niveau d'intérêt qu'elle suscite. Ensuite, pour améliorer la vie privée des utilisateurs, nous proposons des mécanismes d'offuscation permettant de cacher le profil exact des utilisateurs sans trop dégrader la qualité de la recommandation fournie. Enfin, nous explorons un nouveau modèle tirant parti des avantages des systèmes distribués tout en conservant une architecture centralisée. Cette solution hybride et générique permet de démocratiser les systèmes de recommandations en offrant aux fournisseurs de contenu un système de personnalisation à faible coût.

Abstract

The rapid evolution of the web has changed the way information is created, distributed, evaluated and consumed. Users are now at the center of the web and becoming the most prolific content generators. To effectively navigate through the stream of available news, users require tools to efficiently filter the content according to their interests. To receive personalized content, users exploit social networks and recommendation systems using their private data. However, these systems face scalability issues, have difficulties in coping with interest dynamics, and raise a multitude of privacy challenges. In this thesis, we exploit peer-to-peer networks to propose a recommendation system to disseminate news in a personalized manner. Peer-to-peer approaches provide highly-scalable systems and are an interesting alternative to Big brother type companies. However, the absence of any global knowledge calls for collaborative filtering schemes that can cope with partial and dynamic interest profiles. Furthermore, the collaborative filtering schemes must not hurt the privacy of users. The first contribution of this thesis conveys the feasibility of a fully decentralized news recommender. The proposed system constructs an implicit social network based on user profiles that express the opinions of users about the news items they receive. News items are disseminated through a heterogeneous gossip protocol that (1) biases the orientation of the dissemination, and (2) amplifies dissemination based on the level of interest in each news item. Then, we propose obfuscation mechanisms to preserve privacy without sacrificing the quality of the recommendation. Finally, we explore a novel scheme leveraging the power of the distribution in a centralized architecture. This hybrid and generic scheme democratizes personalized systems by providing an online, cost-effective and scalable architecture for content providers at a minimal investment cost.