Personalization of a Peer-to-Peer Television System

Jun Wang[1], Johan Pouwelse[1], Jenneke Fokker[2], Marcel J.T. Reinders[1]

[1]Faculty of Electrical Engineering, Mathematics and Computer Science,
[2]Faculty of Industrial Design Engineering,
Delft University of Technology
Delft, The Netherlands
{jun.wang, j.a.pouwelse, m.j.t.reinders}@tudelft.nl, j.e.fokker@tudelft.nl

Abstract

We introduce *Tribler* a personalized peer-to-peer (P2P) television system. Personalization allows users to browse programs much more efficiently according to their taste. On the other hand, personalization also enables to build social networks that can improve the performance of current P2P systems considerably by increasing content availability, trust and the realization of proper incentives to exchange content. In this paper, we present a novel scheme, called *BuddyCast*, that builds such a social network for a user by exchanging user interest profiles using exploitation and exploration principles. Additionally, we show how the interest of a user in TV programs can be derived from the zapping behavior, thereby avoiding the explicit rating of TV programs. Further, we present how the social network of a user can be used to realize a truly distributed recommendation of TV programs. Finally, we demonstrate a novel user interface for the personalized peer-to-peer television system that encompasses a personalized tag-based navigation to browse the available distributed content. The user interface also visualizes the social network of a user, thereby increasing community feeling which increases trust amongst users and within available content and creates incentives of to exchange content within the community.

1. Peer-to-Peer Television

Television signals have been broadcasted around the world for many decades. More flexibility was introduced with the arrival of the VCR. PVR (personal video recorder) devices such as the TiVo further enhanced the television experience. A PVR enables people to watch television programs they like without the restrictions of broadcast schedules. However, a PVR has limited recording capacity and can only record programs that are available on the local cable system or satellite receiver.

This paper presents a prototype system that goes beyond the existing VCR, PVR, and VoD (Video on Demand) solutions. We believe that amongst others broadband, P2P, and recommendation technology will drastically change the television broadcasting as it exists today. Our operational prototype system called *Tribler* gives people access to all television stations in the world. By exploiting P2P technology, we have created a distribution system for live television as well as sharing of programs recorded days or months ago.

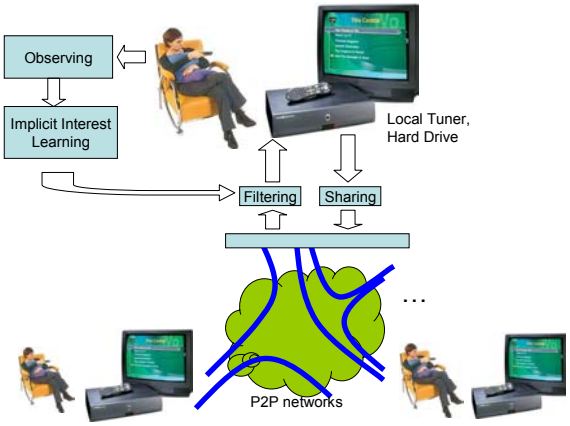


Fig. 1 An illustration of *Tribler*, a personalized P2P television system.

The *Tribler* system is illustrated in Fig. 1. The basic idea is that each user will have a small low-cost set-top box attached to his/her TV to record the local programs from the local tuner. This content is stored on a hard disk and shared with other users (friends) through the *Tribler* P2P software. Each user is then both a program consumer as well as a program provider. *Tribler* implicitly learns the interests of users in TV programs by analyzing their zapping behavior. The system automatically recommends, records, or even downloads programs based on the learned user interest. Connecting millions of set-top boxes in a P2P network will unbolt a wealth of programs, television channels and their archives to people. We believe this will tremendously change the way people watch TV.

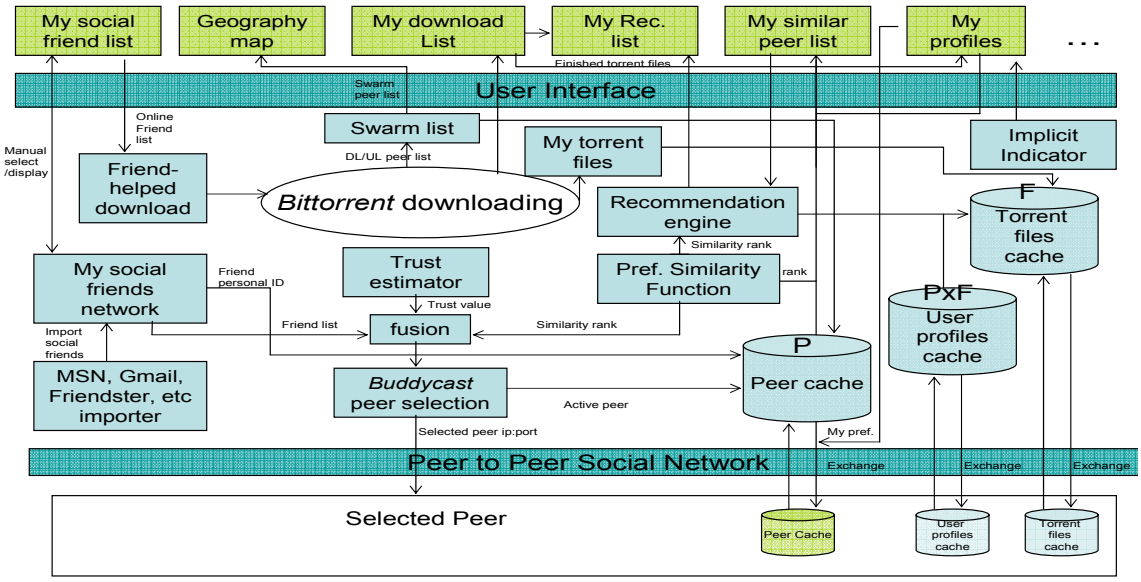


Fig. 2. The system architecture of *Tribler*.

2. P2P TV Personalization

The architecture of the *Tribler* system is shown in Fig. 2 and a detailed description can be found in [Pouwelse et al., 2005]. The key behind the *Tribler* system is that it exploits the prime social phenomenon "kinship fosters cooperation" [Pouwelse et al., 2005]. In other words, similar taste for content can form a foundation for an online community with altruistic behavior. Partly this is realized by building social groups of users that have similar taste captured in user interest profiles.

The user interest profiles within the social groups can also facilitate the prioritization of content for a query user by exploiting recommendation technology. With this information, the available content in the peer-to-peer community can be explored using novel personalized tag-based navigation.

In this paper, we focus on the personalization aspects of the *Tribler* system. Firstly, we review the related work. Secondly, we present a distributed profile exchanger, called *BuddyCast*, which enables the formation of social groups as well as distributed content recommendation (ranking of TV programs). Thirdly, we propose a heuristic scheme that implicitly learns the interest of a user in TV programs from zapping behavior in that way avoiding the need for explicit ratings. Fourthly, we demonstrate a user interface incorporating these personalized aspects, i.e. personalized tag-based browsing as well as visualizing your social group. Finally, we present our experiments to examine the effectiveness of the underlying approaches in the *Tribler* system.

3. Related Work

3.1 Distributed Recommendation

In P2P TV systems, both the users and the supplied programs are widely distributed and change constantly, which makes it difficult to filter and localize content within the P2P network. Thus, an efficient filtering mechanism is required to be able to find suitable content.

We adopt recommendations to help users discover available relevant content in a more natural way. Furthermore, it observes and integrates the interests of a user within the discovery process. Recommender systems propose a similarity measure that expresses the relevance between an item (the content) and the profile of a user. Current recommender systems are mostly based on collaborative filtering, which is a filtering technique that analyzes a rating database of user profiles for similarities between users (user-based) or programs (item-based). Others focus on content-based filtering, which, for instance, based on the EPG data [Ardissono et al., 2004],

The profile information about programs can either be based on ratings (explicit interest functions) or on log-archives (implicit interest functions). Correspondingly, their differences lead to two different approaches of collaborative filtering: rating-based and log-based. The different rating-based approaches are often classified as memory-based [Breese et al., 1998, Herlocker et al., 1999] or model-based [Hafmann 2004].

Few log-based collaborative filtering approaches have been developed thus far. Among them are the item-based Top-N collaborative filtering approach [Deshpande&Karypis 2004] and Amazon's item-based collaborative filtering [Linden et al., 2003]. In previous work, we developed a probabilistic framework that gives a probabilistic justification on the log-based collaborative filtering approaches [Wang et al., 2005] that is also employed in this paper to make TV program recommendation in *Tribler*.

Within the context of P2P networks there is, however, no centralized rating database, thus making it impossible to apply current collaborative filtering approaches. Recently, a few early attempts towards decentralized collaborative filtering have been introduced [Miller et al., 2004, Ali&van Stam 2004]. In [Miller et al., 2004], five architectures are proposed to find and store user rating data to facilitate rating-based recommendation: 1) a central server, 2) random discovery similar to Gnutella, 3) transitive traversal, 4) Distributed Hash Tables (DHT), and 5) secure Blackboard. In [Ali&van Stam 2004], item-to-item recommendation is applied to TiVo (a Personal Video Recorder system) in a client-server architecture. These solutions aggregate the rating data in order to make a recommendation and are independent of any semantic structures of the networks. This inevitably increases the amount of traffic within the network.

Jelasity and van Steen [Jelasity&van Steen, 2002] introduced newscast an epidemic (or gossip) protocol that exploits randomness to disseminate information without keeping any static structures or requiring any sort of administration. Although, these protocols successfully operate dynamic networks, their lack of structure restricts them to perform these services in an efficient way.

In this paper, we propose a novel algorithm, called *BuddyCast*, that, in contrast to newscast, generates a semantic overlay on the epidemic protocols by implicitly clustering peers into social networks. Since social networks have small-world network characteristics the user profiles can be disseminated efficiently. Furthermore, the resulting semantic overlays are also important for the membership management and content discovery, especially for highly dynamic environments with nodes joining and leaving frequently.

3.2 Learning User Interest

Rating-based collaborative filtering requires users to explicitly indicate what they like or do not like [Breese et al., 1998, Herlocker et al.,1999]. For TV recommendation, the rated items could be preferred channels, favorite genres, and hated actors. Previous research [Nichols, 1998, Claypool et al., 2001] has shown that users are unlikely to provide an extensive list of explicit ratings which eventually can seriously degrade the performance of the recommendation. Consquently, the interest of a user should be learned in an implicit way.

In this paper we learn these interests from TV watching habits such as the zapping behavior. For example, zapping away from a program is a hint that the user is not interested, or, alternatively, watching the whole program is an indication that the user liked that show. This mapping, however, is not straightforward. For example, it is also possible that the user likes this program, but another channel is showing an even more interesting program. In that case zapping away is not an indication that the program is not interesting. In this paper we introduce a simple heuristic scheme to learn the user interest implicitly from the zapping behavior.

4. BuddyCast Profile Exchange

BuddyCast generates a semantic overlay on the epidemic protocols by implicitly clustering peers into social networks. It works as follows. Each user maintains a list of its top-N most similar users (a.k.a. taste buddies or social network) along with their current profile lists. To be able to discover new users, each user also maintains a *random cache* to record the top-N most fresh “random” IP addresses.

Periodically, a user connects either to one of its buddies to exchange social networks and current profile list (*exploitation*), or to a new randomly chosen user from the random cache to exchange this information (*exploration*). To prevent reconnecting to a recently visited user in order to maximize the exploration of the social network, every user also maintains a list with the *K* most recently visited users (excluding the taste buddies).

Different with the gossip-based approaches, which only consider exploration (randomly select a user to connect), *Buddycast* algorithm considers

exploration as well as exploitation. Previous study has shown that a set of user profiles is not a random graph and has a certain clustering coefficient [Wang et al., 2006]. That is the probability that two of user A's buddies (top-*N* similar users) will be buddies of one another is greater than the probability that two randomly chosen users will be buddies. Based on this observation, we connect users according to their similarity ranks. The more similar a user, the more chance it gets selected to exchange user profiles. Moreover, to discover new users and prevent network divergence, we also add some randomness to allow other users to have a certain chance to be selected.

To find a good balance between exploitation (making use of small-world characteristics of social networks) and exploration (discovering new worlds), the following procedure is adopted. First, δN random users are chosen, where δ denotes the exploration-to-exploitation ratio and $0 \le \delta N \le$ number of users in the random cache. Then, these random users are joined with the *N* buddies, and a ranked list is created based on the similarity of their profile lists with the profile of the user under consideration. Instead of connecting to the random users to get their profile lists, the random users are assigned the lowest ranks. Then, one user is randomly chosen from this ranked list according to a roulette wheel approach (probabilities proportional to the ranks), which gives taste buddies a higher probability to be selected than random users.

Once a user has been selected, the two caches are updated. In random cache, the IP address is updated to keep it fresh. In the buddy cache, the buddy list of the selected user is joined. The buddy cache is then ranked (according to the similarity with the own profile), and the top-*N* best ranked users are kept. Similarities between profile lists are measured using the Pearson correlation among the binary rating vectors for all the known programs [Breese et al,. 1998].

5. Recommendation by Relevance Model

In the *Tribler* system, TV programs are ranked based on the interest of a user in order to facilitate taste-based navigation. We adopt a probabilistic user-item relevance model to measure the relevance between user interests and TV programs, which intends to answer the following basic question:

"*What is the probability that* this *program is relevant to* this *user, given his or her profile*?"

Let $U \in \{u_1,...,u_K\}$ and $I \in \{i_1,...,i_M\}$ be discrete random variables to represent the users and the TV programs. By building up a relevance model, the relevance rank of a target program i_m towards a target user u_k can be expressed as follows (For detailed information, we refer to [Wang et al., 2005]):

$$Rank_{u_k}(i_m) = \sum_{\forall similar\ u_b: y_b^m=1} \log\left(1+\frac{(1-\lambda)\,freq(u_k,u_b)/freq(u_k)}{\lambda\, freq(u_b)}\right) \quad (1)$$

where $freq(u_k,u_b)$ denotes the co-occurrence of two user profiles, i.e. it equals to the number of programs they both like. $freq(u_b)$ denotes number of programs that user u_b likes. $y_b^m = 1$ means user *b* likes program *m*, which can be learned from user's zapping behavior (will be described in the next section). $\lambda \in [0,1]$ is a smoothing parameter, which is used to balances the maximum likelihood estimation and the background model $freq(u_b)$ (a larger λ means more smoothing from background model).

From the equation, we can see that the relevance rank is calculated based on the opinions of other similar users. For a target user and target program, the rank of their relevance is basically the sum of the target user's co-occurrence with other similar users who have liked the target program ($freq(u_k,u_b)$). The co-occurrence is higher if there are more programs the two users agree upon (express interest in the same program). However, the co-occurrence should be suppressed more when the similar user has liked more programs ($freq(u_b)$), since he or she is less discriminative.

6. Learning From Zapping Behavior

We use the zapping behavior of a user to learn the user interest in the watched TV programs. The zapping behavior of all users is recorded and coupled with the EPG (Electronic Program Guide) data to generate program IDs. In the *Tribler* system different TV programs have different IDs. TV series that consists of a set of episodes, like "Friends" or a general "news" program, get one ID (all episodes get the same ID) to bring more relevance among programs.

For each user u_k the interest in TV program i_m, can be calculated as follows:

(a) Tribler's Open Screen

(b) User Exploitation of a Social Network

(c) Personalized Tag-based Navigation

(d) Content Descriptions of the Recommended Programs

Fig. 3 User Interface of Tribler.

$$x_k^m = \frac{WatchedLength(m,k)}{OnAirLength(m)/\,freq(m)} \qquad (2)$$

WatchedLength(*m*,*k*) denotes the duration that the user u_k has watched program i_m in seconds. *OnAirLength*(*m*) denotes the entire duration in seconds of the program i_m on air (cumulative with respect to episodes or reruns). *Freq*(*m*) denotes the the number of times program i_m has been broadcasted (episodes are considered to be a rerun), in other words *OnAirLength(m)/freq(m)* is the average duration of a 'single' broadcast, e.g. average duration of an episode. This normalization with respect to the number of times a program has been broadcasted is taking into consideration since programs that are frequently broadcasted also have more chance that a user gets to watch it.

Experiments (see Fig. 9) showed that, due to the frequent zapping behaviors of users, a large number of x_k^m 's have very small values (zapping along channels). It is necessary to filter out those small valued x_k^m 's in order to: 1) reduce the amounts of user interest profiles that need to be exchanged, and 2) improve recommendation by excluding these noisy data. Therefore, the user interest values x_k^m are thresholded resulting in binary user interest values:

$$y_k^m = 1 \;\; if \;\; x_k^m > T \;\; \text{and} \;\; y_k^m = 0 \;\; \text{otherwise} \qquad (3)$$

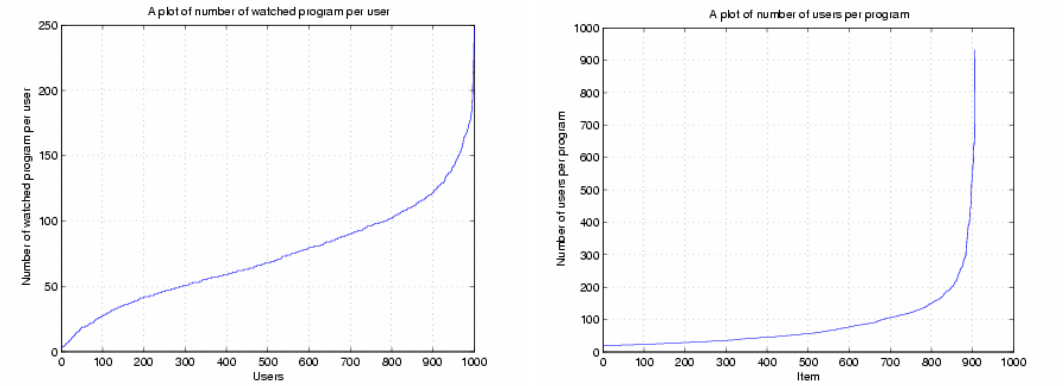
Consequently, y_k^m indicates whether user u_k likes program i_m ($y_k^m = 1$) or not ($y_k^m = 0$).

The optimal threshold T will be obtained through experimentation.

7. Personalized User Interface

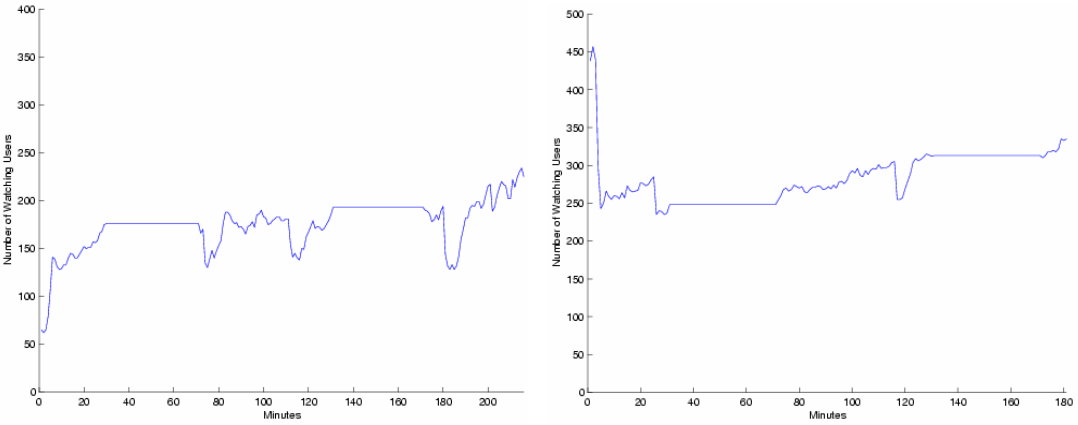
When designing a user interface for a distributed system like *Tribler*, it is important to reach and maintain a critical mass since the users are the decisive factors of the system's success ([Fokker & De Ridder, 2005]). Therefore, several usability aspects have to be dealt with: 1) the wealth and complexity of content, 2) the lack of trust among users, 3) no guarantee of system or content integrity, and 4) the need for voluntary cooperation among users. Here we only addresses and illustrates the first two aspects.

A user is unable to deal with an unlimited number of programs to choose from. Our distributed recommender system helps to filter according to the implicitly learned interests. Subsequently it becomes important to communicate the results in a



(a) Number of Watched Programs Per User **(b) Number of Watching Users Per Programs**

Fig. 4. SKO Data Set of User Actions on Remote Controls.



(a) Film: Live and Let Die (1973)) **(b) Film: Someone she knows (1994)**

Fig. 5 Program Attention.

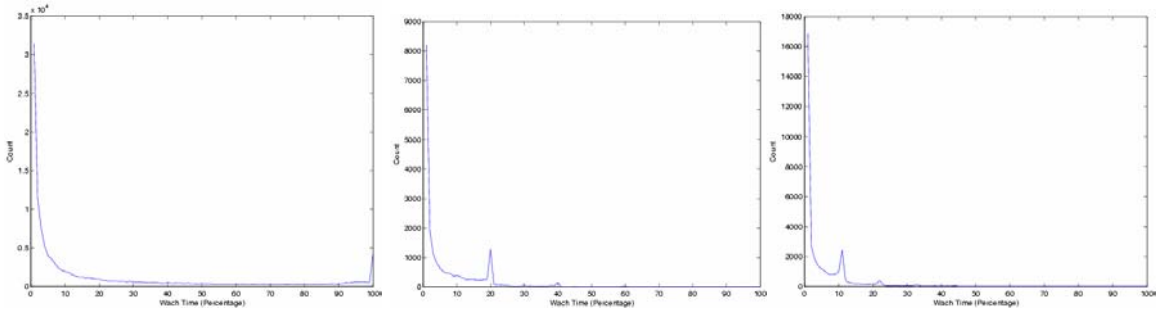
way that makes sense to a user and allows for exploration and exploitation of the available content in spite of the lack of trust amongst users.

In Fig 3. we illustrate our thoughts on a user interface for a decentralized recommender system, as applied in *Tribler*. Fig 3.(a) is *Tribler*'s opening screen. In Fig 3.(b) we show a user's social network in which relations are expressed in social distances: friend, friend-of-a-friend, or taste buddy (which is obtained by running our *Buddycast* algorithm). With this the exploitation of the community is stimulated because users can look into each other's hard disks directly, thus rewarding the risks users take when allowing taste buddies to communicate with them. Fig 3. (c) shows the personalized tag-based navigation, which is a popular way of displaying filtered results or recommendations, as in Flickr.com or CiteULike.org. The font size of each tag reflects its relevance towards user. The relevance rank of each tag can be calculated by summing up all the relevance ranks from its attached programs. This feature incorporates a reflection on the origins, and trustworthiness of the recommended content. We believe this will reduce the uncertainty about the quality and integrity of the programs and lack of trust among users. Moreover it stimulates users to explore new content in a natural way. Fig 3.(d) demonstrates content descriptions of recommended programs. As with the tag-based navigations, it incorporates a reflection on the origins, quality, and integrity. Furthermore, it provides more background information on items, like in IMDb.com.

8. Experiments and Results

We have conducted a set of experiments with the *Tribler* system on a real data set containing the TV zapping behavior of users to address the following questions:

1. What zapping behaviors do we observe and what can be learned from these behaviors to implicitly derive the interest of users in TV programs?
2. How sensitive is the recommendation of TV programs as a function of the user interest threshold T and what is the optimal value



(a) Programs on-air 1 time (b) Programs on-air 5 times (c) Programs on-air 9 times

Fig. 6. Percentage of Watching Time for Programs with Different On-air Times.

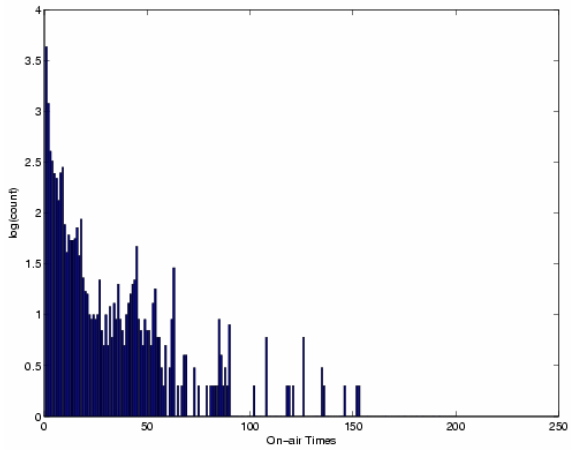


Fig. 7 Program On-air Times during Jan. 1 to Jan 30, 2003.

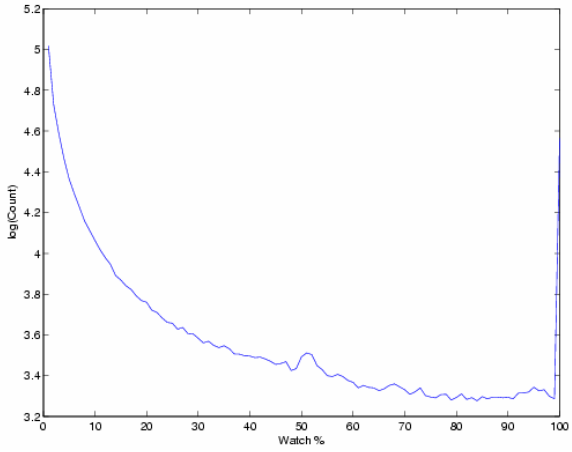
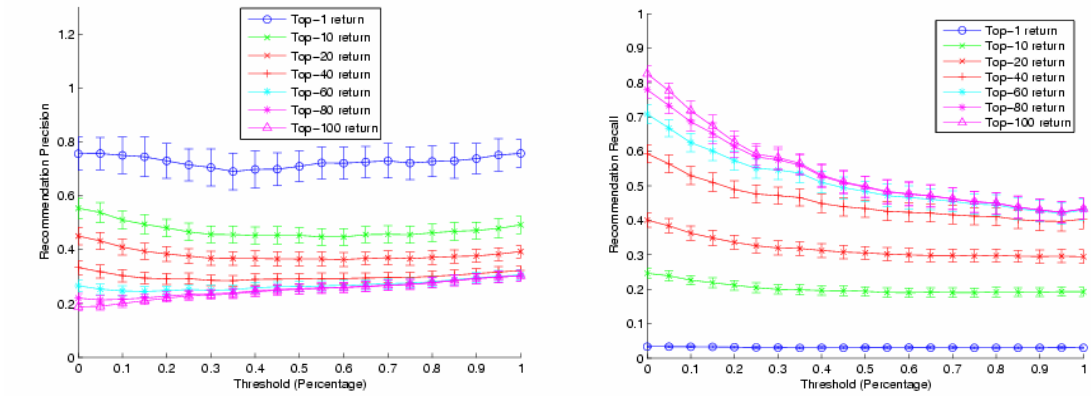


Fig. 8 Normalized Percentage of Watching Time.



(a) Precision of Recommendation **(b) Recall of Recommendation**

Fig. 9. Recommendation Performance V.S. Threshold *T*.

taking into account the efficiency of exchanging interest between users?

3. How efficient is our proposed *BuddyCast* algorithm compared to the newscast algorithm when we want to exchange user interest profiles?

8.1 Data Set

We used a data set that contained the TV zapping behavior of 6000 Dutch users over 19 channels from the SKO foundation[1]. The remote controls actions were recorded from January 1 to January 31, 2003. Some basic characteristics of this data set are shown in Fig. 4. We employed the EPG data set

[1] http://www.kijkonderzoek.nl

obtained from Omroep.nl (an online TV program guide) to find TV program IDs[2]. This resulted in 8578 unique programs and 27179 broadcasting slots over the 19 Channels in that period (this includes reruns and episodes of the samne TV program). Fig 7 shows statistics about the number of times TV programs are broadcasted. For instance, news is broadcasted several times a day. Series have different episodes and are broadcasted for example weekly.

8.2 Observations of the data set

This rich data set can be used to analyze the zapping behavior of users for particular TV programs. In Fig 5 this is shown for a more popular movie, "Live and Let Die" (1973), and a less popluar movie, "Someone she knows" (1994).

For example, when we look at the beginning of the two programs, it clearly shows the difference of the user attention for the less popular film, i..e the number of watching users drops significantly for the first 5 minutes or so. Probably, these users first zapped into the channel to check out the movie and realized that it was not interesting movie for them and zapped away. Contrarily, the number of watching users steadily increasing in the first minutes for the more popular.

Another interesting observation in both figures is that during the whole broadcasting time, there were some intervals of about five to ten minutes, in which the number of watching users dropped. This is because some users left the channel when commercials began and zapped back again when they had supposedly ended.

Fig 6 shows the number of users with respect to their percentages of watching times (*WatchLenght*(*k,m*)/*OnAirlength*(*m*)) for programs with different number of times that they are broadcasted (on-air times of 1, 5 and 9).

This shows clearly two peaks: the larger peak on the left indicates a large number of users who only watched small parts of a program. The second smaller peak on the right indicates that a large number of users watched the whole programs once regardless of the number of times that the program was broadcasted. That is, the right peak happens in 20% of the programs that are broadcasted five times (one fifth), and in 11% of the programs that are broadcasted nine times (1 ninth), etc. There is a third peak which happens in 22% in the programs which are broadcasted nine times. This indicates that there are still a few users who watched the entire program twice, for example to follow a series.

These observations motivated us to normalize the percentage of watching time by the number of broadcastings of a program as explained in Eq. 2, in order to arrive at the measure of interest within a TV program. This normalized percentage is shown in Fig. 8. Now all the second peaks are located at the 100% position.

8.3 Learning the user interest threshold

The threshold level, *T*, above which the normalized percentage of watching time is considered to express interest in a TV program (Eq. (3)) is determined by evaluating the performance of the recommendation for different setting of this threshold.

The recommendation performance is measured by using *precision* and *recall* of a set of test users. Precision measures the proportion of recommended programs that the user truly likes. Recall measures the proportion of the programs that a user truly likes that are recommended. In case of making recommendations, precision seems more important than recall. However, to analyze the behavior of our method, we report both metrics on our experimental results.

Since we lack information on what the users liked, we considered programs that a user watched more than once ($x_{k,m} > 1$) to be programs that the user likes and all other programs as shows that the user does not like. Note that, in this way , only a fraction of the programs that the user *truly* liked are caputered. Therefore, the measured precision *underestimates* the true precision.

For cross-validation, we randomly divided this data set into a training set (80% of the users) and a test set (20% of the users). The training set was used to estimate the model. The test set was used for evaluating the accuracy of the recommendations on the new users, whose user profiles are not in the training set. Results are obtains by averaging 5 different runs of such a random division.

We plotted the performance of recommendations (both precision and recall) against the threshold on the percentage of watching time in Fig 9. We also varied the number of programs returned by the recommender (top-1, 10, 20, 40, 80 or 100 recommended TV programs). Fig. 9.(a) shows that in general, the threshold does not affect the precision too much. For the large number of programs recommended, the precision becomes

[2] http://omroep.nl

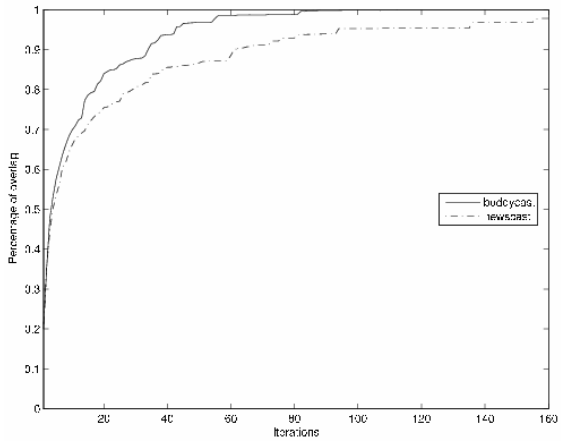


Fig. 10 Convergence of Our Buddycast Algorithm.

slightly better when there is a larger threshold. For larger number of recommended programs, the recall, however, drops for larger threshold values (shown in Fig 9.(b)). Since the threshold does not affect the precision too much, a higher threshold is chosen in order to reduce the length of the user interest profiles to be exchanged within the network. For that reason we have chosen a threshold value of 0.8.

8.4 Convergence behavior of BuddyCast

We have emulated our *BuddyCast* algorithm using a cluster of PCs (the DAS-2 [3] system). The simulated network consisted of 480 users distributed uniformly over 32 nodes. We used the user profiles of 480 users. Each user maintained a list of 10 taste buddies (N=10) and the 10 last visited users (K=10). The system was initialized by giving each user a random other user. The exploration-to- exploitation δ was set to 1.

Fig. 10 compares the convergence of *BuddyCast* to that of newscast (randomly select connecting users, i.e., $\delta \to \infty$). After each update we compared the list of top-N taste buddies with a pre-compiled list of top-N taste buddies generated using all data (centralized approach). In Fig. 10, the percentage of overlap is shown as a function of time (represented by the number of updates). The figure shows that the convergence of Buddycast is much faster than that of the Newscast approach.

9. Conclusions

In this paper, we introduced a personalized peer-to-peer television system called *Tribler*. We focused on the personalization aspects, i.e. 1) the exchange of user interest profiles between users by automatically creating social groups based on the interest of users, 2) learning these user interest profiles from zapping behavior, and 3) a personalized user interface to browse the available content making use of recommendation technology. Experiments on a real data set containing the TV zapping behavior of 6000 Dutch users over 19 channels show that personalization can increase the effectiveness to exchange content and enables to explore the wealth of available TV programs in a peer-to-peer environment.

[3] http://www.cs.vu.nl/das2

References

Ali, K. & van Stam, W., (2004). TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*..

Ardissono, L., Kobsa, A., and Maybury, M. (Ed). (2004). *Personalized Digital Television. Targeting programs to individual users*. Kluwer Academic Publishers.

Breese, J. S., Heckerman, D. & Kadie, C., (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *Conference on Uncertainty in Artificial Intelligence*.

Claypool, M., Waseda, M., Le, P., & Brow, D. C., (2001). Implicit interest indicators. *International Conference on Intelligent User Interfaces.*

Deshpande, M. & Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems.*

Eugster, P.T., Guerraoui, R., Kermarrec, A.M., & Massoulie, L. (2004), From epidemics to distributed computing, *IEEE Computer.* 21(4):341—374.

Fokker, J.E., De Ridder, H. (2005). Technical Report on the Human Side of Cooperating in Decentralized Networks. *Internal report I-Share Deliverable 1.2*, Delft University of Technology. http://www.cs.vu.nl/ishare/public/I-Share-D1.2.pdf

Hofmann, T. (2004). Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems*.

Herlocker, J.L., Konstan, J.A., Borchers, A. & Riedl J. (1999). An algorithmic framework for performing collaborative filtering. *International ACM SIGIR Conference on Research Development on Information Retrieval.*

Jelasity, M & van Steen, M. (2002). Large-Scale Newscast Computing on the Internet. *Internal report IR-503*, Vrije Universiteit, Department of Computer Science.

Linden G., Smith, B., & York J. (2003). Amazon. com recommendations: item-to-item collaborative filtering. IEEE *Internet Computing.*

Miller, B.M., Konstan, J.A., & Riedl, J. (2004) PocketLens: Toward a Personal Recommender System. *ACM Transactions on Information Systems.*

Nichols, D. (1998). Implicit rating and filtering. In *Proceedings of 5th DELOS Workshop on Filtering and Collaborative Filtering*, pages 31-36, ERCIM.

Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D.H.J, Reinders, M.J.T van Steen, M., Sips, HJ. (2005). Chitraka: A social-based Peer-to-Peer system. *International Workshop on Peer-to-Peer Systems (IPTPS'06)*.

Wang, J., de Vries, A.P., & Reinders, M.J.T, (2005). A User-Item Relevance Model for Log-based Collaborative Filtering. *European Conference on Information Retrieval*.

Wang, J., Pouwelse, J., Lagendijk, R., & Reinders, M.R.J., (2006). Distributed Collaborative Filtering for Peer-to-Peer File Sharing Systems, *ACM Symposium on Applied Computing.*