

A gossip-based distributed social networking system

Citation for published version (APA):

Abbas, S. M. A., Pouwelse, J. A., Epema, D. H. J., & Sips, H. J. (2009). A gossip-based distributed social networking system. In *Proceedings of the 18th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'09, Groningen, The Netherlands, June 2-July 1, 2009)* (pp. 93-98). Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/WETICE.2009.30>

DOI:

[10.1109/WETICE.2009.30](https://doi.org/10.1109/WETICE.2009.30)

Document status and date:

Published: 01/01/2009

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

A gossip-based distributed social networking system

S.M.A. Abbas, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips

Department of Computer Science
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands
m.ali.abbas@gmail.com, j.a.pouwelse, d.h.j.epema, h.j.sips@tudelft.nl

Abstract—Social networking web sites, which allow users to create identities and link them to friends who have also created identities, are highly popular. Systems such as Facebook and MySpace utilize a traditional client-server approach to achieve this, which means that all identities and their social links (the entire social network) are stored and administered on central servers. Although this approach supports highly mobile user access - users can log-in from any computer - it also poses high dependence on predefined central server(s), which results in possible exploitation of private data.

In this paper we present an alternative approach, based on gossip protocol, in which we use a completely decentralized peer-to-peer system to create and store the social network. Our system is self-administered and works in a highly transient environment of peer availability. We propose the design and implementation in Tribler of a distributed social networking system that is scalable and robust, allowing users to perform core social networking functions of establishing and removing social links without any requirement for centralized servers or administration.

Index Terms—Peer to Peer, Distributed SNS, Gossip Protocol

I. INTRODUCTION

One of the recent trends seen in the cyber world is the emergence of Social Networking Systems (SNS). An increasing number of people is flocking towards these systems and engaging in new group based social activities. With MySpace¹ and Facebook² being in the top 10 of the most visited websites in the world, a huge potential and affinity of people towards social networking can be seen.

The first major SNS that attracted worldwide attention and proved itself a trailblazer, was Friendster³. It popularized the features that define contemporary social network sites, such as profiles, public testimonials or comments, and public lists of friends [1]. While most dating sites focus on introducing people to strangers with similar interests, Friendster was designed to help friends-of-friends meet, based on the assumption that friends-of-friends would make better romantic partners than would strangers [2]. Overall, the impact of online SNS has been tremendous. However, currently these systems depend on a centralized architecture, and are therefore prone to become the victims of possible exploitation by central authorities.

¹www.myspace.com

²www.facebook.com

³www.friendster.com

Also, being a centralized system, they are highly dependent on centralized entities with complete authority.

In this paper, we propose a design and initial implementation of a decentralized SNS based on a gossip protocol, due to its light-weight nature and also high scalability characteristic. Our system establishes friendship links among peers under dynamic conditions of peer availability. As has been observed in distributed systems, be it structured or unstructured, that the rate at which the peers join and leave the system, or in other words, peers churn rate, is very high [3]. We have developed our system which establishes friendship links among them in such a dynamic environment. The possible peers which may become friends are discovered in a gossip fashion. Our system is self administered and does not depend upon any central entity, which means that all of the social network is handled by the peers themselves. The notion of friendship link establishment in our system is the basic building block of forming the SNS. On top of this structure, many applications can be developed. Currently, in Tribler, cooperative downloading [4] is one of the applications using friendship links. It actually makes use of idle bandwidth of ones friends to boost ones download performance.

The implementation of our solution has been done in Tribler, which is a peer-to-peer client. Tribler [5], which is a Bittorrent based P2P client. It facilitates users to find the right content, and then share it with others.

The structure of this paper is as follows. In Section 2, we discuss the functionalities of our SNS provided to the the users, along with the concepts already part of Tribler. In Section 3, our detailed design is presented. Section 4 discusses the evaluation of our work and its reliability with the help of some experiments we carried out with the deployed system. Possible attacks and their prevention in our system are discussed in Section 5. Future work is presented in Section 6 and related work in Section 7. The paper ends with the conclusion in Section 8.

II. REQUIREMENTS OF OUR SYSTEM

In this section, we will discuss requirements of our solution. First, a set of functionalities provided to the users are discussed in Section II-A. After that, Section II-B highlights the basic concepts of Tribler which are relevant to our solution.

A. Functionalities

In this section, the functionalities provided to the users of our SNS are going to be listed. Below, the terms user and peer are used interchangeably. In our SNS, the peer who initiates a friendship request to another peer is known as the *source peer*, and the peer for whom this request is intended is known as the *target peer*. The functionalities provided to the users are the following:

- *Adding new friends* In order to build a social circle, a peer can request other peers discovered by the underlying peer sampling service (PSS), which are potential candidates of friends, to become their friends. The source peer requests the target peer to become its friend. The target peer has to reply to this request, and if the reply is positive, the peers become friends.
- *Removing friends* The source peer removes the target peer from its friends list. Also it requests the target peer to remove it from its list.
- *Maintaining status of friends* The system must keep peers up-to-date about the online status of their friends.

B. Tribler

The implementation of our SNS has been done in Tribler, which is a Bittorrent based file-sharing client. A little background on important and relevant concepts of Tribler need to be highlighted.

In Tribler, peers have a permanent identifier (*PermID*), which is based upon public-private key pairs. A peer, the challenger, can challenge another peer, the challengee, for its identity by generating a large random number. The challengee encrypts it with its private key, and then the challenger decrypts the result with the public key of the challengee. If the result of this decryption is the same as the random number generated, the authentication succeeds.

Tribler has an epidemic protocol called *BuddyCast* for peer and content discovery service. In Buddycast, peers exchange messages with random peers (exploration) and semantically close peers called Taste Buddies (exploitation). After a pairwise exchange, both the involved peers merge their lists of peers and then rank it according to their preference list similarities with this list. They both retain only the top N best ranked peers. The notion of Taste Buddies, or semantically close peers, helps to reduce the randomness of peers, which eventually leads to better search results for content searching. Peers take care not to contact the same peer for the next 4 hours.

The contextual information based on prior communication through Buddycast among peers is stored in Tribler in a local database known as the *Mega-Cache*. It stores information about peers, torrents, and preferences (list of recent downloads). This is then used by Tribler to gossip, calculate similarities, and recommend torrents. The Mega-Cache is filled by means of Buddycast service.

III. DETAILED DESIGN

In this section, we are going to explain how the establishment of friendship link between the source and the target peers is developed in a very dynamic environment in which peers may go offline at any time. In Section III-A, we are going to highlight the basic mechanism of friendship link establishment, based on the request-reply concept, between the source and the target peers. In Section III-B, we are going to present the underlying retry mechanism and also the notion of the *helpers* with their role. Based on the target and the source peer availability, we will present the friendship link establishment scenarios in Section III-C. At the end, in Section IV, we will share how peers find out their friends' online status.

A. Basic request-reply protocol

For establishing a friendship link, the mechanism follows the request-reply notion. The source peer initiates it by sending a friendship request to the target peer. Target peer then takes its decision over it by accepting or denying the friendship request. And then finally, target peers reply is sent back to the source peer. If the reply is positive, both the source and the target peer become friends.

B. Unavailability of the peers

In order to deal with the unavailability of both the source and the target peer, we have designed two mechanisms, which work for both friendship requests and friendship replies. There two mechanism are listed below:

1) *Retry*: If the target peer is not online, the source peer will retry to connect to it after every five minutes, in case the target peer comes online. Similarly for receiving reply from the target peer, if the source peer is not online, or unconnectable for some reason, the same retry mechanism is adopted by the target peer to dispatch its response to the source peer. The initial retry time interval of *5 minutes* is increased to *24 hours*, after one day has passed since the friendship request/reply was initiated. After a week of unsuccessful delivery of request or reply, all pending friendship messages (requests and replies) are dropped from the source and the target peers.

In order to increase the chances of contacting the other peer, both the source and the target peers save locally unsuccessful delivery of messages, i.e., the pending messages (requests and replies), in case they are going offline. On their next session, both of them read and then dispatch them. We present the underlying retry mechanism shown in Figure 1.

2) *Helpers*: To increase the chances of establishing friendship link between the target peer and the source peer, we have introduced the concept of *helpers*. Helpers are online friends and taste buddies of both the source peer and the target peer. When the source peer is unable to connect to the target peer for requesting friendship link establishment, it dispatches its friendship request to these helpers. Helpers then try to contact the target peer every five minutes. These helpers as stated above, are also used by the target peer for forwarding its friendship reply to the source peer, in

case it is unable to contact it. Helpers, just like the source and the target peer, also save the unsuccessful friendship requests/replies locally when they are going offline. And on their next startup, they try to deliver them to the intended peer.

C. Scenarios of establishing friendship links

Depending upon the availability of the source and the target peer, we distinguished different scenarios for establishing friendship link between them. Note that these scenarios only show the friendship request part. The reply part follows the same scenarios.

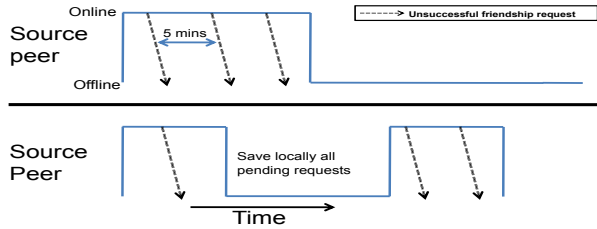


Fig. 1. Friendship request retry mechanism

The possible scenarios of friendship link establishment between the source and the target peer are the following:

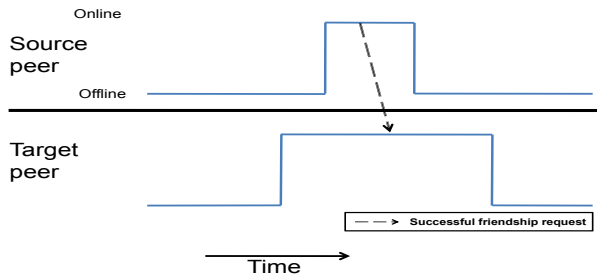


Fig. 2. Friendship request scenario 1: Both the source peer and the target peer are online.

- **Case 1: Both the source and the target peers are online.** Source peer directly sends the friend request to the target Peer. Depending upon the target peers response, it is added, or deleted from the source peers friends list. Figure 2 shows the interaction between the source and the target peer.
- **Case 2: The source peer is online, but the target peer is not.** Source peer after a failed attempt to connect to the target peer, would try the retry mechanism mentioned

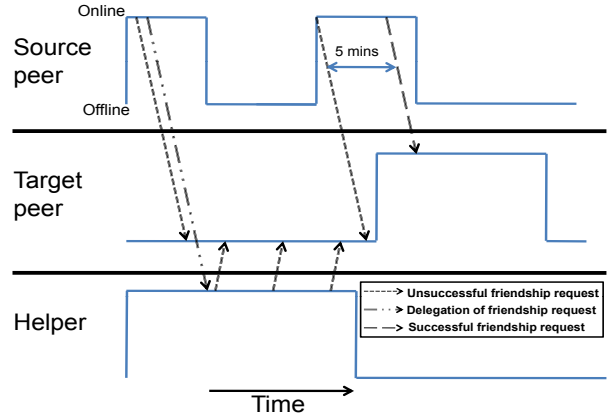


Fig. 3. Friendship request scenario 2: Both the source peer and the helpers try to contact the target peer

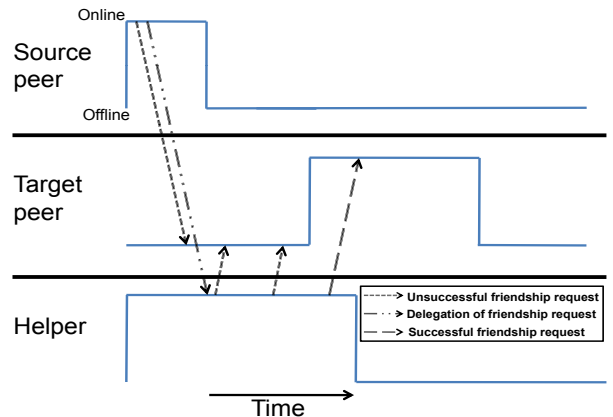


Fig. 4. Friendship request scenario 3: On behalf of the source peer, the helpers relay the friendship request to the target peer.

above, involving both itself and the helpers. This scenario is shown in Figure 3.

- **Case 3: The source peer has gone offline after initiating the friendship request, but the target peer is online.** Since the source peer can not connect to the target peer, it dispatches the friendship request to its helpers. The helpers then connect to the target peer and forward the friendship request to it. This interaction can be seen in Figure 4.

Figure 5 shows the screenshot of friendship establishment request in Tribler.

IV. ONLINE STATUS MANAGEMENT

As soon as a peer starts his client, the underlying service, BuddyCast, starts contacting his list of his friends to see if they are online or not. Batch by batch all friends of the peer are contacted. On a successful connection establishment with them, their status is set to *online*, otherwise *offline*. This

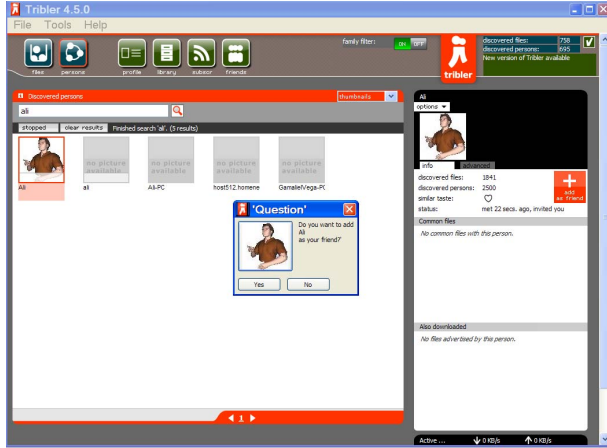


Fig. 5. Screenshot of friendship link establishment

process continues in the background till the peer remains online, so that the up-to-date status of all of his friends could be displayed. In short, the peer's online status is set to online if a successful connection is made to him.

V. EXPERIMENTS

In this section, we will show the success rate of our system with the help of reliability experiments. Success over here means that the source peer indeed get the reply from the target peer on its friendship link establishment request. Regardless of it being positive or negative. Our system was deployed in Tribler 4.5 release on November 11th, 2008. To record friendship establishment related statistics of our SNS, we have developed a crawler, which is a specialized P2P client, and is being run on one of our servers at TU Delft. According to the crawlers statistics, the overall population of our system was 535 till 20th November, 2008. That means, the collected data is of 10 days. Based on the list of peers supplied by the underlying PSS (Buddycast, in this case), this crawler connects with every peer which comes online, and asks it to supply all the friendship requests it has made so far. In case of stumbling upon the same peer after a while, it asks for new friendship requests since the last encounter and also updated friendship records which have now received the replies. All clients save their friendship requests and replies record in their local database. The analysis in this section has been made on this retrieved data. In the first result in Section V-A, we present all the friendship requests made by all the peers we contacted. And also, it shows how many of them were successful. In Section V-B, we show for each of the friendship link establishment request, how much time was taken for its response.

A. Number of friendship link establishment requests

Over a period of 10 days, in total, there were 588 friendship link establishment requests made by 132 peers. Out of those

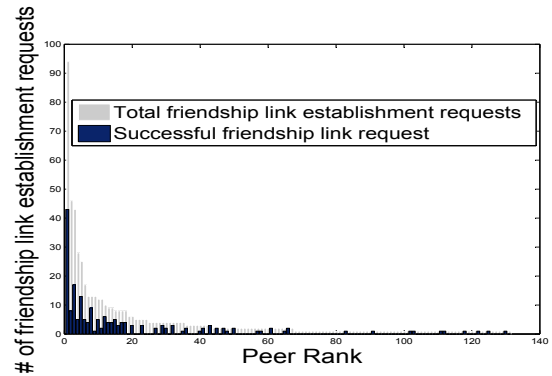


Fig. 6. The number of total and successful friendship link establishments

requests, 191 were successful, resulting in a success rate of 32 percent. There are several reasons for such a success rate. The target peers may not have come online after the source peer initiated the request. Even if they did, they were not online at the same time as source or helpers were. Or lastly, the request has expired. As mentioned in Section III-B1, after a certain time period, i.e., a week, all pending requests are expired. Figure 6 shows this result. White bars represent total requests made by a peer, and grey bars represent how many of them were successful. In order to know what is the actual successful rate, we have parsed log files on our super peers used by Tribler, which record data of all peers (tribler clients) when they come online to get the list of fresh peers through Buddycast. Out of those 588 requests, the target peers mentioned in 298 requests, never came online throughout the crawling phase. Also, in addition to that, in 39 requests, the involved target peers were not seen in the duration of the source peer initiating the friendship request till the end of expiry of the request, i.e, after 7 days. That means, out of 588 requests, only 251 represent the total friendship requests. Keeping this figure in mind, now the overall success rate becomes 76 percent.

B. Total time taken for receiving friendship replies

The total time used in requesting and then getting a reply on friendship link establishment can be seen in Figure 7. We show the results of both friendship link establishment requests with both positive and negative replies. For all the 191 requests, it shows the histogram of time in minutes taken to make them successful. On average, almost 128 minutes were taken to get the response.

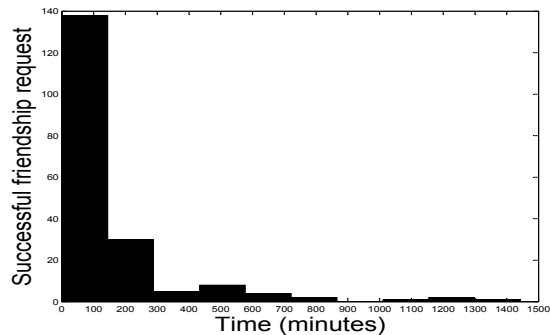


Fig. 7. Histogram of the time taken for each successful friendship requests

VI. POSSIBLE ATTACKS PREVENTION

In the current design, it is possible for malicious peers to target and subvert the system. There are mainly two possible attacks, which are Distributed Denial of Service (DDOS) and special DDOS, *man in the middle* attack. In order to thwart such potential attacks, we have established certain safeguards which we shall detail below along with them.

DDOS is a type of attack where a peer is asked by a huge number of other peers for some service. The motive behind this attack is to overload a peer so that even legitimate peers are unable to access it and get its service. For the DDOS attack, we restrict a user, who is running the client from the exe, or binaries, to make at maximum 10 friends per day. We can not, of course, overcome this problem if a user has modified our code. This can be accomplished fairly easily as all the friendship requests will be recorded by the system. The system shall place an upper limit of 10 friendship requests per day on the user. This means, all subsequent requests shall be ignored.

In the *man in the middle* attack, the *helper* tries to overload a peer, or a group of peers, with a huge number of illegitimate friendship requests. To counter this, we have devised another solution to overcome this attack by incorporating the use of signed requests. That means, the peer who initiates the request (source peer) first signs it with its private key. This would allow the receiver (target peer) to determine that it is indeed from the source peer. Since only one instance of Tribler client can run on a single machine, no malicious peer can fake or develop multiple instances, and thus multiple identities.

VII. FUTURE WORK

Here we present two possible extensions of our work. First is the *mobile identity* and the second one is *real-life contacts search* extension.

Binding a person to his peer identity which is independent of his IP address, computer and also current location, so that he can regain its social network no matter from where he joins the system. We will be focussing on the problem facing a peer whenever it wants to rejoin the P2P system, either after it having lost its data, or having changed its computer, or even its location. We call this the *mobile identity* problem of the peers. We want to enable such a peer to easily regain its social network. It would ask the system for its social network by supplying its credentials, i.e., username and password (we assume that it is only known to it). And then the system will gossip around and try to find its friends, who had previously stored its social network. Its request would be directed to its friends, who would eventually help it regain its social network.

We would also like to extend our current SNS by incorporating searching and then establishing friendship links with one's real-life friends. After getting the contacts list from a peer's associated email service, we will carry out search of its real friends. In case they happen to already exist. The process would first search locally in the peer's mega-cache, and then expands its search by contacting highly connected peers (peers with a large social circle), and then ask them for the peer's friends. Once they have been found, our retry mechanism would come into the play and then try to establish friendship links with them.

VIII. RELATED WORK

A number of SNSs exist, which are of distributed nature, but none of them are purely distributed. There are always some central entity involved in one way or another. Skype, with over 13 million users, is a super-peer P2P based chat and voip system. Being a closed source system, it is difficult to analyze, but some analysis has been done [6]. Skype includes only one centralized server for user authentication and to keep usernames unique throughout the system. Each Skype client locally saves information about its friends (Buddy List). It also contains a Host Cache (HC) which bears a list of Super Nodes (SN), in which, at least one SN has to be valid in order to Skype function. All peers with public IP can potentially become an SN. Searching a user is also possible via Global Index, provided that the target user has logged in the last 72 hours.

Maze [7] uses a social network to communicate and discover files. It uses a centralized ticketing server known as Ticket Grant Server (TGS), which issues tickets to all peers to identify them. This ticket is then served as a form of legitimate communication/transaction between peers. The ticket is only valid for a single communication. Also, Maze uses another centralized server called Heartbeat server which, apart from holding a directory of peers, also checks the online status of each of them. In social maze, where friends can help discover

new peers via their friends, it can run without involving this server; however, the TGS would still be required.

P-Grid [8] uses a structured network. For dealing with identity of peers, keeping dynamicity of P2P in mind, it, just like Tribler, establishes unique ID locally, which is generated via a hash function of current date and time, IP address, and a large random number. In case of change of IP of a node - either because it has re-joined the system or DHCP has assigned a new IP, so called replicators help identify that node. Also, based on structured network, Symphony [9] and SPROUT [10] target routing strategy based on the social links. SPROUT [10] defines a trust model based on social distance (in terms of hops) between two nodes. The farther the node in the social network is, the lower the trust would be. For a particular key k , it forwards the query to one of the online friends whose node ID is closest to k . Look-ahead is also possible with distance of friends-of-friends id to the target key k taking into account.

SybilGuard [11] presents a solution to minimize Sybil attacks. It exploits social network by stating that Sybil nodes can be detected and ignored, since they will not have many trusted links with genuine or trusted nodes. It, however, does not describe how the social network would actually be bootstrapped, and is left as a future work.

Based on semantic routing, Borch in his work, Social P2P [12] forms groups in a peer-to-peer fashion. It searches content, and then form implicit groups, leading to the formation of a social network. Also, depending on manual user preferences - over a period of time - a node can become closer to those who are nearer to its interests.

For dealing with an estimation how many peers' information should locally be saved for gossiping - without knowing the actual number of peers in the system - SCAMP [13] adjusts local partial view of peer membership accordingly.

IX. CONCLUSIONS

This paper described a completely decentralized, self-administered, light-weight and scalable Social Networking System (SNS). The peers, which are potential friends, are discovered through gossip based protocol. To overcome the dynamicity of peer availability in P2P systems, we have demonstrated our mechanisms which establish friendship links between peers in such a transient environment. Current implementation and deployment have been done in Tribler P2P client, but it can run on any gossip based P2P client. We have also carried out reliability experiments to show the behaviour of our SNS in real-life scenarios. Our data for the experiments have been collected with our deployed SNS under Tribler 4.5 release. To thwart unwanted and malicious attacks of DDOS and *man in the middle attack*, we have developed safeguards to restrict users to overload the system, and to use signed friendship requests. Two further extensions of our work have also been presented.

REFERENCES

- [1] D. Boyd, "Why youth (heart) social network sites: The role of networked publics in teenage social life," *Identity. MacArthur Foundation* [- \[2\] D. Boyd and N. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.
 - \[3\] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, *Handling Churn in a DHT*. Computer Science Division, University of California, 2003.
 - \[4\] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, "2fast : Collaborative downloads in p2p networks," *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pp. 23–30, Sept. 2006.
 - \[5\] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips, "Tribler: a social-based peer-to-peer system," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, p. 127, 2008.
 - \[6\] S. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," *Arxiv preprint cs.NI/0412017*, 2004.
 - \[7\] C. Hua, Y. Mao, H. Jinqiang, D. Haiqing, and L. Xiaoming, "Maze: A social peer-to-peer network," *E-Commerce Technology for Dynamic E-Business, IEEE Intl. Conf. on*, vol. 0, pp. 290–293, 2004.
 - \[8\] K. Aberer, A. Datta, and M. Hauswirth, "Efficient, self-contained handling of identity in peer-to-peer systems," *IEEE Trans. on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 858–869, 2004.
 - \[9\] G. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, pp. 127–140, 2003.
 - \[10\] S. Marti, P. Ganesan, and H. Garcia-Molina, "Dht routing using social links," in *The 3rd Intl. Workshop on Peer-to-Peer Systems*, San Diego, CA, USA.
 - \[11\] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proc. of the 2006 conf. on Applications, technologies, architectures, and protocols for computer communications*, 2006, pp. 267–278.
 - \[12\] N. Borch, "Social peer-to-peer for social people," in *The Intl. Conf. on Internet Technologies and Applications*, 2005.
 - \[13\] A. Ganesh, A. Kermerrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. on Computers*, pp. 139–149, 2003.](http://www.danah.org/papers/[March 15, 2007], 2007.</div><div data-bbox=)