Supporting Collaboration and Creativity Through Mobile P2P Computing

Adam Wierzbicki, Anwitaman Datta, Łukasz Żaczek, and Krzysztof Rzadca

1 Introduction

Among many potential applications of mobile P2P systems, collaboration applications are among the most prominent. Examples of applications such as Groove (although not intended for mobile networks), collaboration tools for disaster recovery (the WORKPAD project), and Skype's collaboration extensions, all demonstrate the potential of P2P collaborative applications. Yet, the development of such applications for mobile P2P systems is still difficult because of the lack of middleware.

The largest potential for mobile P2P applications today lies in the use of mobile phone platforms, especially of phones for the 3G (UMTS) network, or general-purpose platforms like Ultra-Mobile PCs (UMPCs). The recent increase in the capabilities of these devices has been dramatic. Today, it is possible to develop a variety of applications for mobile phones or other devices that would share resources of their users', including information that is available to the particular users only. However, in mobile P2P environments, not only is it impossible to used advanced

The work presented in this article is partially supported by A*Star SERC GRANT NO: 072 134 0055.

Adam Wierzbicki
Polish Japanese Institute of Information Technology, Warsaw, Poland,
e-mail: adamw@pjwstk.edu.pl

Anwitaman Datta
Nanyang Technological University, Nanyang Avenue, Singapore,
e-mail: anwitaman@ntu.edu.sg

Łukasz Żaczek
Polish Japanese Institute of Information Technology, Warsaw, Poland,
e-mail: lzaczek@pjwstk.edu.pl

Krzysztof Rzadca
Nanyang Technological University, Nanyang Avenue, Singapore, e-mail: krz@pjwstk.edu.pl

X. Shen et al. (eds.), *Handbook of Peer-to-Peer Networking,*
DOI 10.1007/978-0-387-09751-0_50,

middleware based on the Service Oriented Architecture, but it becomes difficult to reuse the most developed platforms for P2P systems, such as JXTA or the Microsoft P2P framework.

P2P applications for mobile devices can be developed today with the support of the 3G network's infrastructure. Such applications are not using a pure Peer-to-Peer computing model, as they require specialized servers of the 3G infrastructure to perform user and resource location. Yet, the sharing of resources by such applications can be done in a purely P2P manner. For 3G mobile phones, the use of SIP and the IP Multimedia Subsystem (IMS) can enable the development of such hybrid P2P applications. Petri Pöyhönen, Vice President of Nokia Networks, has chosen the following words to describe the purpose of IMS: "we have no practical mechanism to engage another application-rich terminal in a *peer-to-peer* session. (...) *there is immense value in sharing with peers*. (...) We will be sharing real-time video (see what I can see), an MP3-coded music stream, a whiteboard to present objects, and we will be exchanging real-time data." This quote indicates that like SIP, IMS is seen as a middleware to develop P2P applications. Yet, a fully distributed P2P application would be independent of the operator's IMS infrastructure, and would consequently be able to operate also in an environment where the UMTS network is not available. The challenge therefore is not just in the support of P2P services in a mobile environment, but rather to support them in a *heterogeneous mobile and fixed environment*, enabling collaborative work for nomadic teams with or without infrastructure support, preferably without significant disruption of services or decrease of service capabilities.

The developing P2PSIP standard is a candidate for supporting such P2P applications for mobile devices. Pure P2P applications that are used for collaboration can be modeled using three layers: the *application layer* which implements application-specific logic, the *collaboration middleware layer* that provides several supporting functions for the application layer and may use P2PSIP, and the *P2P overlay layer* that implements the most basic function required by the collaboration middleware layer, including reliability and security functions.

Assuming such a layered decomposition, the question remains what are the crucial services that should be included in a collaboration middleware that would support a wide range of collaborative applications? A second question could be: is the decomposition between the collaboration middleware and P2P overlay layers? This decomposition is, after all, purely logical, since all of the logic would be implemented in the application layer of the OSI protocol stack. In this chapter, we shall demonstrate that a vertical integration of the two layers could result in several advantages. These advantages are mostly due to the fact that the collaboration middleware layer can supply the P2P overlay layer with important information: the *implicit social network created by mining the metadata and statistics produced by the collaboration applications* (or by the function calls of the collaborative middleware). Using social network information in routing and network management can not only enhance the performance of routing in mobile P2P environments (particularly in Delay Tolerant Networks), but also allows to satisfy additional constraints on routing

(such as trust requirements). Other P2P overlay functions, such as publish-subscribe operations, can also be supported by social network information.

The first contribution of this chapter lies in a systematic design of a middleware for P2P collaborative application in a heterogeneous, mobile/fixed environment. We show that the SIP protocol could be an important substrate in the implementation of such a middleware, and analyze which middleware functions could be supported by SIP (and which must be implemented separately). We give a comprehensive overview of the emerging standard of P2PSIP. Another contribution bases on the concept of vertical integration of the collaborative middleware and P2P overlay layers. We review related work that proposed solutions that were vertically integrated to some extent; then, we describe a new concept of SocialCircle, a P2P overlay that is much more closely integrated with the implicit social network provided by the collaboration middleware layer. We give preliminary results of experiments with a prototype of SocialCircle.

The chapter is organized as follows: the first section discusses the functionality of the collaboration middleware, with an emphasis on the most challenging performance and security problems that can require new research. This section also describes the information that can be provided by the collaboration middleware to the underlying P2P overlay layer for cross-layer optimization. The section includes an overview of related work on P2P middleware for mobile collaborative applications.

In the next section, we describe protocols and frameworks that can be used to implement the desired collaboration middleware functions: SIP and P2PSIP. The section discusses the current state-of-the-art in this area, identifying the most difficult research problems.

The last section concerns how the P2P overlay layer can use information provided by a vertically integrated collaboration middleware layer in order to optimize its operation, and how the overlay can be specially designed to better support collaborative applications. This section gives an overview of some recent relevant developments the area, and indicating the way ahead and open research issues to realize a P2P infrastructure tailor-made to support collaborative work in a mobile environment, and also using meta-information from the upper layers in order to make the P2P infrastructure more resilient and efficient.

2 Applications for Collaboration and Creativity Support

Research and development on Computer Supported Collaborative Work (CSCW) is an old area (there exist mature, commercial technologies such as LotusNotes and Microsoft Outlook) that has recently accelerated due to the use of the Internet. Wikipedia, SourceForge, CVS and SVN, iGoogle are examples of successful applications of Internet-based CSCW. Use of the P2P computing model in this area is yet in its infancy. Still, applications such as Groove, Skype and many others show high potential. In 2005, the first International Workshop on Collaborative Peer-to-Peer

Information Systems (COPS 2005) has taken place in Linkoping, Sweden. Since then, the workshop is held yearly.

Creativity support is a younger research area that has been developed mostly by psychologists and management scientists. Computer science has been applied to creativity support since the beginning of the XXI-st century. The first applications, like Oasen, Serious Creativity, Mindmanager, Maxthink, Axon, Groupsystems&Groupintelligence, focus on using knowledge management for idea generation and recording, as well as on enhancing creativity through computer-supported collaboration [17, 19, 33]. Many of these applications apply advanced visualization and human-computer interaction technology, and do not have a collaborative character, but focus on the individual creative process. On the other hand, creativity support is strongly connected to CSCW, since it requires support for debating, brainstorming, and group decision making.

A step towards the development of Peer-to-peer groupware has been the technology of Groove Networks [1]. It concentrates on providing support for collaboration of a team on a project. Groove offers workspaces for aggregation of documents, messages, and application-specific data, that are synchronized among all peers in a workspace; Groove also offers chat sessions, notepads, a calendar, and a file archive. Groove also has a comprehensive security model. Yet, Groove does not offer knowledge management, semantic social networks and functions for creativity support; also, while it uses the P2P computing model, it has not been designed for a mobile environment. Groove users must have access to an infrastructure of "relay servers", which means that the system is not well suited to nomadic use.

There has not been much research on collaboration and creativity support on mobile devices. Shark [32] supports management, synchronization and exchange of knowledge in mobile user groups. Shark uses a semantic meta-data hierarchy for routing of messages in its P2P overlay network. However, Shark does not posses features that support creative processes; Shark is also not suitable for teamwork. The reason why Shark does not have these capabilities is that it has been designed for a low-bandwidth mobile ad-hoc network (Bluetooth).

Two interesting examples of creativity support software that could be ported to the P2P model and, in the process, extended with collaboration capabilities, are Freemind and JabRef. Freemind is an open-source application that enables visual mindmapping. JabRef enables the management of bibliographic references using the BibTex standard, as well as searches and automatic acquisition from popular Web-based bibliographic databases such as CiteSeer or DBLP. Both FreeMind and JabRef are really useful for support of the individual creative process; the applications are opensource (written in Java), and could be used on mobile devices (because of low resource requirements). Yet, both applications would benefit from an extension that would enable collaborative work. FreeMind could enable co-authoring of mindmaps, and JabRef could support sharing, searching and recommendation of references. Both of these functions could be simply supported through a central server (like the popular service of Bibsonomy), but supporting them in a heterogeneous, fixed/mobile environment for nomadic teams of users requires the use of the P2P

computing model. The development of both extensions is underway in the mTeam project [3].

We believe that the development of applications for collaboration and creativity support on mobile P2P systems is still difficult because of the lack of middleware. While recent advances in technology of mobile devices and wireless networks enable the use of such technology for collaboration and creativity support, little is known about supporting such applications without centralized servers. For this reason, development of P2P middleware capable of working on mobile devices is a crucial technological and research challenge.

3 Collaboration Middleware Functionality

3.1 General-Purpose Mobile P2P Middleware

Several research and development projects have proposed general-purpose middleware for developing arbitrary applications on mobile devices using the P2P model. Most of these efforts aim to provide a common framework for the rapid development of applications.

Mobile Chedar [5] is an extension to the Chedar framework that provides an API to application developers. Mobile Chedar uses Bluetooth and J2ME, and uses a superpeer model. A special gateway node (a PC with fixed Internet connectivity) manages the other Mobile Chedar nodes and provides Internet access.

The oldest and best-known open-source P2P middleware is JXTA [25]. JXME [23] is a lightweight implementation of JXTA for mobile devices using the J2ME platform. JXME is aimed at devices that are too resource-constrained in order to support the full JXTA protocol; rather, JXME relies on a relay proxy to provide full JXTA functionality.

JMobiPeer [7] is a middleware for J2ME that offers interoperability with JXTA. It provides support for discovery and group management.

Proem [24] aims to support collaborative applications, but is in fact a general-purpose P2P middleware for mobile devices in an ad-hoc network. Proem is implemented in Java Standard Edition, and can therefore be run only on powerful PDAs or UMPCs.

MOBY [20] enables P2P exchange of services and data, dynamic service location and client mapping to achieve higher performance and reliability. MOBY relies on JINI (which is not compatible with J2ME).

While all of the general-purpose middleware discussed above can be used to support collaborative applications, these proposals *do not address the specific problems of collaboration and creativity support*. Another drawback of the proposed generic approaches is that most of them *are tightly integrated with a specific P2P overlay network*. This makes it impossible to port the generic middleware to various types of

overlay networks. Only JXTA supports, to some extent, different overlay variants, although it mostly relies on a hybrid superpeer architecture.

In this part of the chapter, we shall describe functional requirements for a middleware that would be dedicated to the support of collaborative work. We shall assume that these functions are independent of the specific P2P overlay network, although their operation could influence how the overlay works through vertical integration. Later, we shall discuss how elements of this middleware could be implemented.

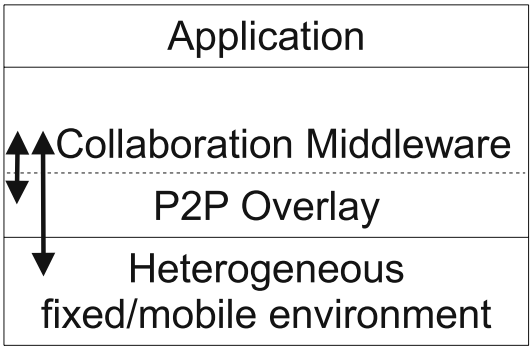


Fig. 1 Vertically integrated collaboration middleware

3.2 Overview of Middleware Functions

The purpose of developing middleware is the support of application development. In the process of middleware design, it is therefore necessary to identify and describe functionality that should be useful for supported applications – in our case, collaborative applications. We have based the functions presented below on a review of the functionality of collaborative applications. It is usually hard to become convinced that the chosen middleware functions are indeed comprehensive, until the middleware has been used to create several applications. Therefore, a larger set of functions must be chosen that can support a variety of applications. However, at the design stage it is possible to reduce certain middleware functions to others, and to choose the most generic versions of required functionality.

An important problem is that the chosen functions of a middleware should satisfy several requirements of efficiency or security. For the purpose of vertically integrated middleware, it also becomes significant what information can be obtained from middleware function execution for the purpose of optimizing lower layers and how this information can be stored, updated, and communicated.

An abstract design of the collaborative middleware is presented on Fig. 1. The interface between the P2P Overlay and the Collaboration Middleware layers should be generic and will be discussed in the next sections. Functions of the Collaboration Middleware include:

3.2.1 Session Management

This function enables the initiating, modifying, and ending of sessions of multiple users. Session management requires discovery of the participants, and therefore the use of the P2P overlay layer routing function. A session describes the communicated media, including information about the encoding and transport method, between all participants of the session (each participant can receive a different set of media streams). Therefore, session establishment and modification requires signaling.

3.2.2 Conference Management

This function enables the establishing and updating of conference state. The state can include the conference membership. Also, the conference state can include floor control (allowing only one user to speak at a time), or another form of control (for example, moderation). Conferences can have several modes of establishment: for example, dial-out mode, where a single user invites all other conference participants, or join-in mode, were new users join the conference whenever they want.

The conference management function requires use of the event management function (an event is that a user has joined a conference). The conference itself uses the P2P overlay layer multicasting function for the delivery of conference media.

The session and conference management functions can be the source of usage statistics that can determine the level of acquaintance between persons to be used in a *social network*. Each user could maintain these statistics about his own communication, enabling the creation of an *egocentric social network*; on request, users could share their egocentric networks with other, authorized or trusted users. In this way, a global view of the social network could be constructed by trusted superpeers.

3.2.3 Data Synchronization

Data used by arbitrary other services, or by the application layer, will need to be synchronized among a group of authorized devices (that can belong to a single or multiple users). This synchronization can have several models – from a strong synchronization (all changes are propagated to all devices, WYSIWIS – What You See Is What I See), to a relaxed synchronization that allows different versions of the data to exist at the same time on different devices (concurrent work). Two special cases of data synchronization are: document synchronization and workspace synchronization.

3.2.4 Data Sharing

Unlike in data synchronization were the data is intended to produce a single version from all participants (in relaxed synchronization, data could have different versions,

but the intention is to produce a single version in the end), some forms of data should be shared only with some (authorized and interested) users and are not intended to produce a single version. Furthermore, the data itself could be controlled by an authorization policy that governs their distribution. Using this function, a user of a collaborative application can make certain information available to other, interested users who can copy and distribute this information without needing to synchronize it with the original version.

3.2.5 Presence

State and profiles of users in the system should be made available to other, authorized users. The user (or device) profile can include available services or resources. Overall, the presence service would be responsible for the maintenance of all available context information about the user, including location information (if available). For example, consider a context that describes what kind of device is currently in use by the participant; this kind of information would be part of the service profile maintained by the presence service, and could be obtained automatically. The presence service requires the use of the overlay layer publish-subscribe service.

Presence information can be the source of important statistics. Users who have a high availability (are nearly always present) are good candidates for superpeers, and can be considered more reliable for various other functions (network management, storage). The availability statistics can be maintained by the peer who stores presence information about another peer in the overlay and communicated to trusted superpeers on demand.

3.2.6 Workspace Awareness

When users form a group that works in a shared workspace (or on one document), other users must be aware (to the extent possible) of what the others are doing, in order to enable coordination and planning of work (for example, who is currently editing a certain document). This function is a specialized form of data synchronization.

3.2.7 Group Management

Group management is the function of creating, modifying, deleting of groups of users, including the possibility of establishing hierarchical groups. The group can be working in a shared workspace. Groups can be of many kinds: they can be open (anyone can join), closed (only the group owner can join new members).

3.2.8 Access Control

This function is crucial for the enforcement of authorization policies in a collaborative environment that allows users to share information. The policies can be intended to protect sensitive information, a user's privacy, or business policies. An

access control method for the collaborative application would have to support not only the first access to shared information, but all subsequent sharing of the provided information. Access control can be based on various models, but all of them require the use of an authentication service that should be provided by the P2P overlay layer.

3.2.9 Trust Management

The function of trust management is required in open environments, were the users must depend on the actions of others and are uncertain about the outcome of such interactions. The purpose of trust management is decision support in such circumstances. For example, consider dynamic teams that include participants who do not know each other. In this case, participants can rely on information from trust management for making decisions that require dependence on unknown team members. In return, after gaining knowledge about the new team members, participants will give feedback to the trust management system that can be used to update trust or distrust. Note that trust or distrust need to be maintained in a context that can, in the simplest form, be represented by a value out of a finite set of contexts.

3.2.10 Event Management

This service allows the subscription to events, notification of subscribed users about events, and publishing of events. It uses the P2P overlay layer publish-subscribe service. In particular, the event management service should also include an ordering of events (especially important for synchronization).

3.2.11 End-to-End Security Functions

While overlays may provide some hop-by-hop security functions, these may not be sufficient for the purposes of the collaborative application. For that reason, end-to-end confidentiality should be provided by the middleware. In addition, authentication between the communicating peers (session participants) is required.

3.3 Required P2P Overlay Functions

3.3.1 Routing

Routing of information to a destination (a key that can be a user or service identifier).

3.3.2 Publish-Subscribe

This service allows the subscription to an arbitrary topic, notification of subscribed users about changes to the topic, and publishing changes to a topic.

3.3.3 Multicast

A special case of the publish-subscribe service where a user subscribes to a group. Notifications are any messages from the group, and publishing changes means sending a message.

3.3.4 Overlay Adaptation Functions

These functions enable vertical integration by receiving data from the collaboration middleware layer and adapting overlay structure using the received information. Mostly, this means adapting the local view of the social network that is used to support the P2P overlay network.

3.4 Research Issues Related to Middleware Function Design

The proposed middleware functions should satisfy strong requirements of performance, scalability and security. Furthermore, in a mobile P2P environment their design will have to differ from designs used in fixed P2P networks. These considerations point to the conclusion that new research issues will need to be solved in the middleware design and implementation. It is not the purpose of this chapter to present solutions of the described problems, but merely to identify the most difficult and important issues for future research.

The *session management* service, and therefore the routing service, should have a low call setup latency. This is especially difficult in a mobile P2P environment. Therefore, this requirement poses new research challenges; a special kind of routing may be required for session setup messages. Another requirement is the reliability of session management; how often would a session establishment fail because routing failures due to churn?

The *event management* and *presence service*, and consequently the overlay publish-subscribe service will be required to optimize the load on the notifier, network traffic for notification, and delay of notification delivery. In addition, the requirement of delay tolerance or reliability can be imposed on these services; for example, if a peer departs from the network, it may need to receive all missed notifications after it joins the network again.

In a mobile P2P environment, the *data synchronization* service would have to operate in the concurrent work mode. A question arises how this service should implement a distributed transaction that would produce a single version out of the existing, concurrent versions. This may be achieved through a pre-commit stage (a "code freeze"), but the duration of this pre-commit stage during high churn and long peer disconnections may need to be high. Also, the data synchronization function may need to implement work scheduling decisions that have been made by the team, allowing only certain changes to be made by particular users.

The *data sharing* function bases on the information delivery mechanism of the P2P network. The multicast or publish-subscribe service may be used for that purpose. These services would have to satisfy efficiency and reliability requirements. Also, the associated function of *access control* would have ensure that the data is sharing in agreement with a specified policy. The fundamental problem here is that when a user shares data with another, authorized user, the receiving user could share the data with unauthorized users. How to control this second (and further) degree of sharing is a difficult research problem.

The *trust management* service will need to use some expression of trust context, which may be based on a semantic description of the problem that the collaboration is trying to solve. Also, the kind of feedback information used by the trust management system, and the algorithms used to calculate trust or distrust measures are the subject of ongoing research. For example, the feedback information could be based on edit history or on an explicit evaluation of team partners.

4 SIP and P2PSIP

The Session Initiation Protocol (SIP) and its extensions already implement a number of the functions required by the collaboration middleware (although not all). Moreover, SIP is implemented and available to programmers on most modern mobile phones, and SIP is specially treated by many firewalls and NATs. Since many P2P protocols today are implemented on top of an application layer protocol (such as HTTP), these observations can lead to the conclusion that the collaboration middleware for mobile P2P environments should be implemented using SIP. The question remains how, and to what extent, SIP can be used to implement all required middleware services.

In this section, SIP is briefly introduced (with a focus on its relevant services). In the next part of the section we discuss how the dependence on server infrastructure can be removed through the use of P2PSIP. Research and standardization of this extension is continually advancing. Yet, for the purpose of a collaboration middleware, all required middleware services would have to be implemented in a distributed, P2P environment. This requires a new design effort, and raises issues of security and performance.

4.1 The Session Initiation Protocol

The Session Initiation Protocol (SIP) [30] is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. SIP sessions can be used for any kind of application data, and SIP does not standardize or attempt to control session contents. Thus, the first required service of the collaboration middleware that can be implemented using SIP is the Session Management service.

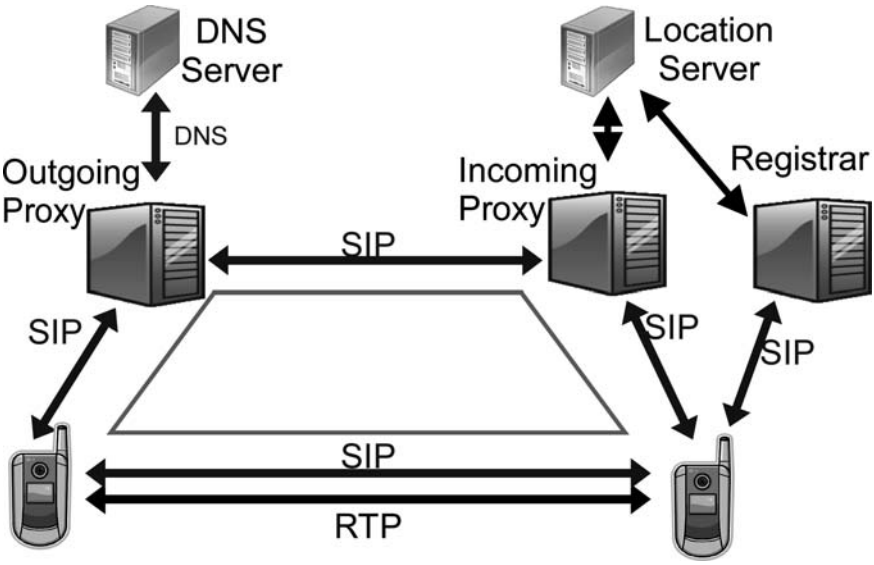


Fig. 2 SIP signaling using proxy and registrar servers

SIP introduces its own addressing of users (or services), with the intension of achieving independence of the user's location in the network (IP address). Moreover, SIP session management services allow to search for users at several locations during session initiation, and to specify redirection policies that can depend on the user's current state and will redirect a session to another location.

SIP can use either TCP or UDP for message transport; due to the possibility of using UDP, acknowledgement messages are required and a three-way handshake is used during session initiation. SIP uses an infrastructure of servers for the resolution of its addresses and for the control of the session. SIP servers can be of four kinds (the proposed distinctions are logical, and server roles can be combined): registrar servers, proxy server, redirect servers and location servers. Another type of SIP server is a gateway proxy server. Proxy servers can be thought of as analogous to local DNS servers that take over responsibility for SIP address resolution. Proxies can also work as gateways to other networks or domains, and SIP signaling can be forwarded by arbitrary chains of proxies. Proxy servers can also implement a variety of services, such as lists of missed calls, call forwarding or screening. Proxy servers may also implement call admission control and report usage. For the purpose of some service, proxy servers may operate in a stateful model and maintain state information for the duration of session initiation (not during the session).

The registrar server is analogous to an authoritative DNS server that contains the definite information required for SIP address resolution. A SIP user must register with a registrar server in order to be able to receive sessions.

Figure 2 shows the SIP signaling flow between two user agents, two proxy servers and a registrar server. Note that the user agents exchange signaling messages only

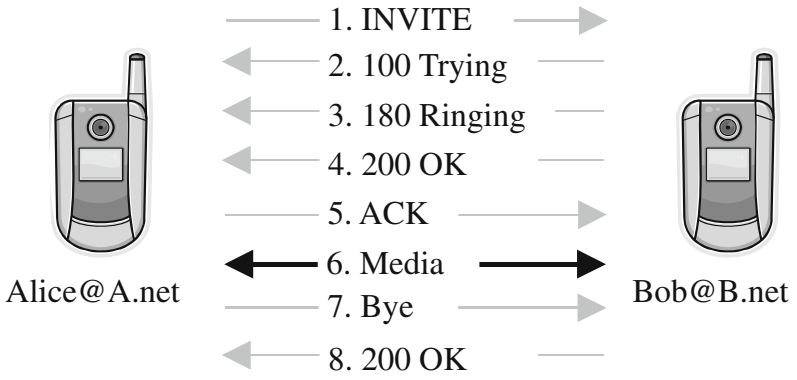


Fig. 3 SIP message flow for simple session

Table 1 SIP methods

Basic methods	Extended methods
INVITE	INFO
ACK	NOTIFY
BYE	PRACK
REGISTER	PUBLISH
OPTIONS	SUBSCRIBE
	UPDATE
	MESSAGE

with their local proxies. Note also that when the roles of a proxy server, registrar server and location server are distinct, the proxy server does not signal the registrar server directly using SIP; rather, both of them interact with the location server using a non-standardized protocol. The signaling flow is simplified when proxy, registrar and location servers are integrated.

The SIP protocol is a text-based protocol that resembles HTTP. It can use several methods, which form two sets. Table 1 summarizes the basic and extended methods of SIP. SIP messages use mandatory headers that resemble mail messages, and can include a MIME body and Session Description Protocol (SDP) information [18] (Fig. 4).

Figure 3 shows a basic SIP message flow during a simple session initiation. Note that the response codes of message exchanges resemble HTTP. Figure 2 shows the signaling message flow during a SIP registration procedure.

For the implementation of the Session Management service of the collaborative middleware, SIP provides comprehensive negotiation functions that allow users to agree on received media formats and transport, as well as updating session state by adding or removing participants, or changing their session parameters. Such

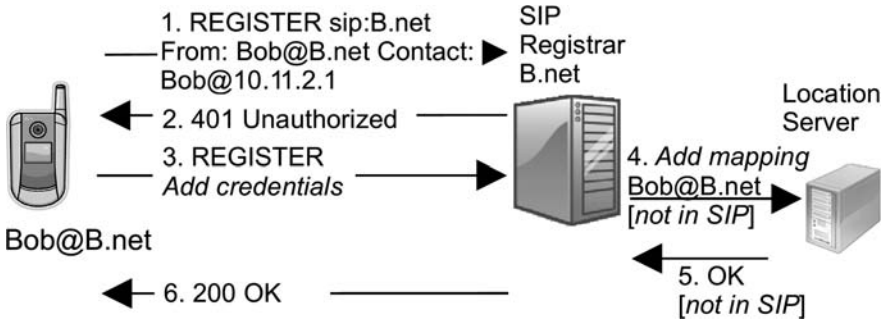


Fig. 4 SIP registration

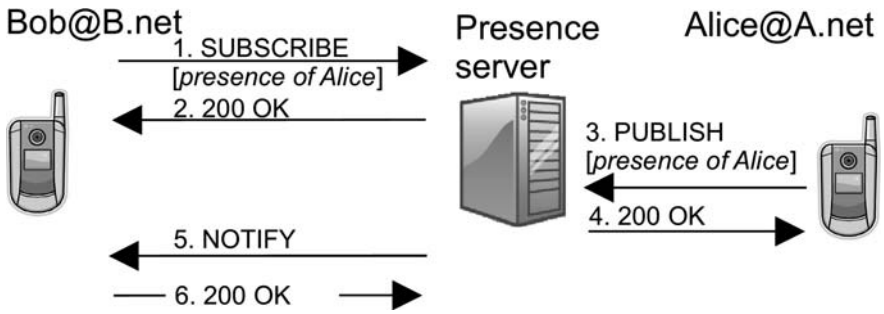


Fig. 5 SIP presence service

functionality is essential for session management and using SIP seems the easiest way to implement a comprehensive Session Management service.

SIP has been extended with several functions that allow to implement various services of the collaboration middleware. To a limited extent, the Conference Control service can be implemented using SIP; however, SIP does not have features for floor control in a conference. The Data Sharing service can also be partially implemented, since SIP has an extension for file sharing (SIPPING) [15]. SIP has further extensions that support challenge-response authentication based on a shared secret (although a SIP proxy can also implement RADIUS authentication).

SIP provides the most comprehensive support for the Presence and Event Notification services [27]. Figure 5 shows the SIP message flow that implements a presence services. Note that the implementation is based on a server and uses three new SIP methods: SUBSCRIBE that allows an agent to indicate that it is interested in receiving information about an event (in this case, the presence of another user), PUBLISH that is used to inform the server about new information, and NOTIFY that is used by the server to notify a client. The figure is simplified because it does not show a notification message that automatically follows the response to the SUBSCRIBE message.

Presence in SIP can be extended to include information about changes in buddy lists, by adding authorization policies that control who can receive presence

information (using the XCAP standard) [28]. Another extension concerns presence data; using the Presence Information Data Format (PIDF) or the Rich Presence Extensions to PIDF (RPID) [31, 36] it is possible to express detailed status of users, using information that could be automatically extracted from calendars. PIDF and RPID are XML dialects. The SIP extensions for presence and instant messaging [11] are developed by the SIMPLE IETF working group [29].

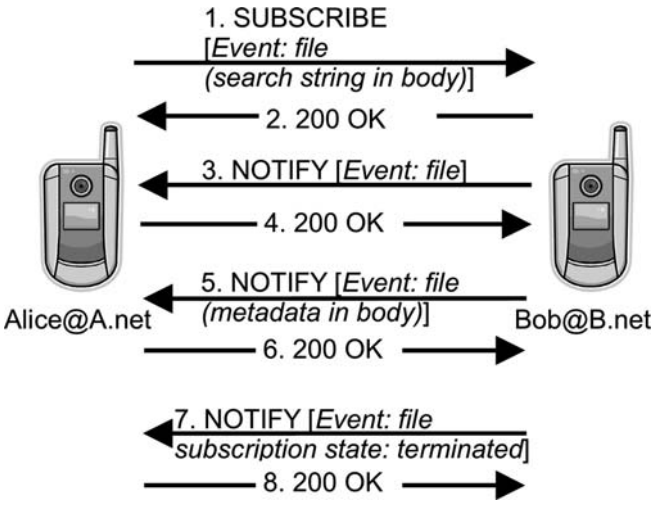


Fig. 6 The SIP event notification

Presence is a special case of SIP's event notification framework that is shown on Fig. 6. SIP allows the subscription to arbitrary events, and the notification about these events will be sent to all subscribed users. This design of the event notification service can create severe performance and scalability problems.

4.2 P2PSIP

SIP can be considered today as a stable and mature technology that has seen wide adoption in Internet-based application. Additionally, the choice of SIP as a signaling protocol for IMS and its consequent implementation by 3G mobile phones has further increased the importance of SIP as a signaling protocol. In the previous sections, we have shown how SIP can be used to implement several services of the collaboration middleware. Additionally, IMS is a powerful middleware that facilitates easier development of many types of applications, although it is not specially designed for collaboration services. On the other hand, applications developed using IMS will depend on the operator's 3G network. In an environment that does not have access to the 3G network, IMS applications cannot function.

The developing P2PSIP standard aims to enable the use of SIP without the dependence on a server infrastructure. The first step in this effort is to distribute the SIP registrar servers using a P2P overlay. Following this, other SIP functions such as publish-subscribe for event management and presence can also be distributed, aiming to create a P2P framework that would support full SIP functionality (a kind of P2P-IMS). The full set of requirements for a P2PSIP system would include [34]:

1. **Zero configuration**. The system should be able to automatically configure itself, e.g., detecting NAT and firewall settings, discovering neighboring peers and performing initial registration.
2. **Heterogeneous nodes**: It should be able to adapt to available resources and distinguish between peers with different capacity and availability constraints. This favors the distinction between nodes and super-nodes as in Kazaa.
3. **Efficient lookup**: The system should be able to execute a lookup of a SIP identifier efficiently, which can be achieved using a structured P2P network.
4. **Advanced services**: The system should support advanced telephony services such as offline voice messaging, multi-party conferencing, call transfer and call forwarding as well as advanced Internet services such as presence and instant messaging. For the support of collaboration, the system should support all collaboration middleware functions described in the first section.
5. **Interoperability**: It should easily integrate with existing protocols and IP telephony infrastructure.

The conceptual model of a P2PSIP system [8] consists of *peers that are coupled with SIP agents* such as user agents, proxies, redirect or gateway servers. In other words, it is possible to think of peers as executing a separate functionality but being coupled with ordinary SIP entities. All peers should be able to participate in the P2P overlay network using the *P2PSIP peer protocol*. However, there may exist SIP user agents that wish to use the P2PSIP system, but are not coupled with a peer capable of participating in the overlay. These peers can use the *P2PSIP client protocol* to contact proxy peers that participate in the overlay. Finally, it is also possible for a SIP agent that does not understand the P2PSIP protocols to communicate with the P2PSIP system directly using SIP.

A high-level overview of the steps that a peer has to carry out in a P2PSIP system is as follows:

1. In order to join an existing overlay, the peer needs to locate a P2PSIP peer that already participates in the overlay, which is referred to as a bootstrap node. IP addresses of bootstrap nodes can be obtained in many ways: from a cached list, using multicast discovery, using manual configuration or public bootstrap nodes.
2. After bootstrapping, the peer should authenticate itself, traverse NAT and firewalls, and register itself in the overlay (or register as a P2PSIP client to some P2PSIP proxy peer). Authentication methods or NAT traversal are not defined yet.
3. After joining, the peer uses the P2PSIP peer protocol to search for other peers or resources, share own resources or insert new resources in the system. Resources can be SIP identifier-IP address mappings, context information, meta-

data, or files. There are several candidates for a P2PSIP peer protocol, all of them are structured P2P overlay networks.

4. After P2PSIP peers discover other peers, they can initiate SIP sessions or send other SIP messages independently of the P2PSIP peer protocol, but possibly using resources of the P2PSIP system such as proxy or gateway peers (also needed for NAT traversal).

At a first glance, the task of distributing the functions of SIP registrar servers that implement a simple directory function seems easy. Structured P2P overlays are capable of providing directory functions. Yet, there is to date no standard of a P2P overlay. Also, P2P traffic often struggles to pass NATs and firewalls, often hiding using another application layer protocol. These considerations raise the first question concerning the design of P2PSIP. Should P2PSIP use a P2P overlay as an external component (requiring SIP clients to implement another protocol and raising issues of choosing the best overlay and of firewall traversal), or can a P2P overlay be designed to form part of SIP? In other words, can a P2P overlay be created and maintained using the SIP protocol itself?

For these reasons, one of the first proposals of a P2PSIP system advocated the use of the SIP protocol as a signaling protocol for overlay setup and management [8, 9]. This proposal specified the mechanism for user registration and location in a P2PSIP overlay. The overlay was created using a modified SIP REGISTER message that could be used to join the overlay and to update overlay neighbors. The proposed approach was implemented using Chord as the P2PSIP overlay. Special SIP identifiers were used to identify peer nodes (a SIP identifier included a Chord node identifier and could specify whether the nodes' IP address was known).

4.2.1 P2PP

The other approach to developing a P2PSIP system requires the use of an additional protocol for overlay management. A current proposal of the P2PSIP working group advocates the development of a new standard for a general-purpose protocol for P2P overlay network management that would also offer the basic services required by P2PSIP [6]. This protocol, called simply the P2P Protocol (P2PP), is independent of the overlay routing and allows the construction of arbitrary overlays (structured or unstructured).

Figure 7 shows the reference architecture of P2PP [6]. The protocol provides an API to the SIP user agent or other applications. The API consists of three separate interfaces: the *service interface* provides functions to join or leave the overlay, functions for routing table maintenance and updates. The *data interface* provides functions to insert, lookup or delete an object by its key. The *diagnostic interface* provides functions for gathering statistics, like response time or relay bandwidth of a peer.

The P2PP protocol does not specify how the routing and network management of the overlay works. In fact, it is possible for the overlay to be structured or unstructured, and for the structured overlay to use any routing scheme. However, this

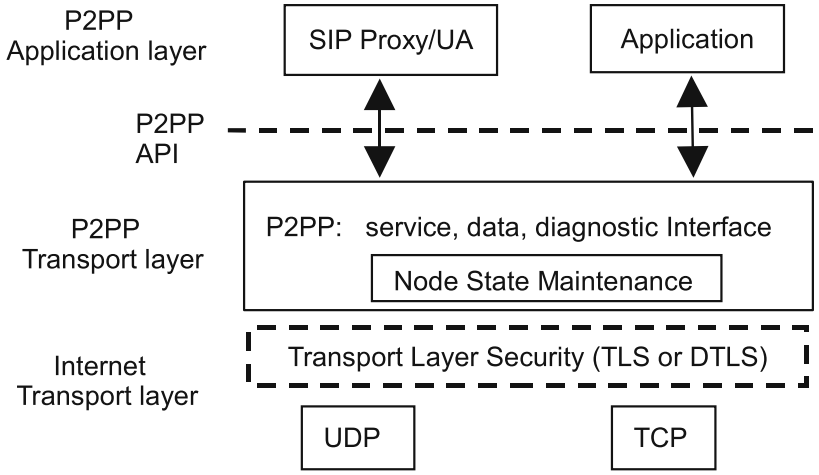


Fig. 7 P2PP reference architecture

does not imply that P2PP specifies just the interface part of the overlay. The protocol specifies how peers maintain state. The P2PP node state contains the following information:

- The P2P overlay routing algorithm (can be any)
- Overlay ID
- Hash algorithm (if any)
- Routing type (iterative or recursive)
- Routing table (table of other peers in overlay)
- Neighbor table (table of immediate neighbors)
- Both tables include for any peer: Peer ID, IP address, RTT, Uptime
- Number of neighbors (sum of routing and neighbor table sizes)
- Resource table (objects the peer is responsible for)
- Replicated resource table (replicated objects)

Overlay routing can be iterative or recursive, which needs to be decided at the time of overlay creation. Peers can issue *parallel iterative routing requests*; then, one copy of the request needs to be designated as primary. Requests use TTLs. The protocol also supports request transactions, which can add reliability to requests if UDP is used for transport. Reliability is provided as a hop-by-hop mechanism. An interesting feature of P2PP is that it can combine the use of TCP and UDP within one overlay. With recursive routing, TCP and UDP can be combined using hop-by-hop reliability for UDP.

P2PP supports the use of security using hop-by-hop TLS or DTLS. An authentication server can be used that issues certificates to users. The P2PP specification describes adversary models and the protocol is protected against some routing and storage threats, as well as replay attacks.

P2PP is a candidate for the generic interface between the Collaboration Middleware and P2P Overlay layers, described on Fig. 1. The reason for this choice is that P2PP is by itself generic, allowing the implementation of arbitrary overlays while keeping the same protocol and service interfaces. However, P2PP does not support all of the functions of the P2P Overlay layer, described in the previous section. In particular, P2PP does not support publish-subscribe or multicast. Of course, the P2PP interfaces do not include overlay adaptation functions. Research on an adaptation of P2PP to support all of the required functions is continuing in the mTeam project [3]. The P2PP interface has already been extended to support publish-subscribe and multicast – a proposal of revision of the standard is forthcoming.

4.3 A Case Study: SharedMind

SharedMind is a collaborative version of FreeMind [2], the most popular open-source mind mapping software. Our motivation is to enable the group to work on a shared mind map. Each member of the group can interactively add, delete or modify nodes in the mind map. These modifications are seen almost in real time by the whole group. If the network connection between groups' members is broken, the members can continue to work independently (or in subgroups, that have connectivity). After the connection is restored, the files are merged. SharedMind is an ongoing work that prototypes and tests the design assumptions of the collaborative middleware.

The original FreeMind follows a standard, model-view-controller architecture. The mind map is represented internally as a tree. Nodes of the tree correspond to the nodes of the mind-map. The text of a node and its style information are stored as properties of the node. The persistent version of the tree is stored as an xml file. User's modifications (such as adding a node, editing node's description or its attributes) are represented as "commands". These commands are executed on the model by the controller.

In our implementation, each member of the group has a local copy of the shared data (the mind map edited by the group). The main issue in SharedMind is to provide seamless *data synchronization* of changes made to local copies. Other functions defined in Section 3.2, especially those related to the security and group management, will be implemented by the middleware and do not require substantial modifications to the FreeMind code. We use P2PP as a generic interface to a P2P overlay. All the members subscribe to a common publish-subscribe topic. Whenever one of the members modify her local copy, a message containing the description of the modification (an xml string containing the marshalled action) is published on (sent to) this shared topic. The middleware propagates the modifications, by the standard subscribe mechanism, to the other members of the group. A member, after having received such a remote modification, unmarshalls it, checks for possible conflicts and, in case there are no, applies it on her local copy.

In this collaborative editing scenario, two main types of conflicts may occur. Firstly, a remote modification may concern a node that is currently edited by the local user. Secondly, modifications may be concurrent. In this case, local editing has finished and the notification message is sent, but the message has not reached the author of the remote modification, before she sent her message. Both types of conflicts are easy to detect. The first one by a simple state variable that stores the currently edited node. The second one, by using logical vector clocks for messages. However, both conflicts have to be resolved by the user, on the level of the UI.

We also plan to support delay-tolerant editing, in which group members continue to work on shared data, even if they do not have any network connection. Delay-tolerant editing relies on presence (provided by the middleware), versioning and checkpointing (specific to FreeMind's data structures, thus implemented over FreeMind's code). The group chooses a random number (using a bully selection algorithm) that becomes the current version number of the data. The presence function of the collaboration middleware periodically checks whether all the group's members are reachable. Group composition changes when a member disconnects, or when a persistent network failure divides the group into subgroups. In such event, each member locally reverts to the last common version of the shared data (at the moment of the last successful presence protocol invocation), and checkpoints this version with the previously elected version number. Afterwards, a member re-applies the modifications that followed and continues to work with the reachable part of the group (electing the new version number). After the connection is restored, the re-united group should produce a common version of the data from the modifications done in each subgroup. The presence mechanism notifies group's members. Then, each subgroup elects a member responsible for merging the data into a one, common version (a *merger*). Mergers exchange lists of version numbers. Then, each merger determines the last common version number (the last number that appears on both lists). Each merger produces a list of modifications its subgroup made over the common version, using the previously checkpointed revision. These lists of modifications are then send to the other merger. Next, mergers merge all the non-conficting changes and resolve conflicts (on the level of the UI). Then, an elected merger applies changes that were made by other, non-merger group's members during the process of merging. This process may also involve possible conflict resolution. Finally, the elected merger's data is considered as the common group's data.

To conclude, the complete version of SharedMind will use most of the middleware functions (Section 3.2). Security-related functions are mainly independent of the original code, and thus do not require substantial modifications. Data synchronization turns out to be most complex to implement, as the data structure is tightly coupled with the original code. Currently, data is synchronized on the application layer (and not by the middleware), using functions provided by the collaboration middleware and the P2PP protocol, such as presence, publish-subscribe and event management. In future, we plan to provide generic synchronizing data structures at the middleware layer, such as a tree or a table (a collection of records). These data structures will be extended by application developers to host application-specific data.

4.4 Research Issues Related to SIP and P2PSIP

SIP and IMS are both client-server systems that have performance and scalability problems. In particular, the presence and event management services of SIP are prone to severe performance problems when they have many users. Another problem is the intolerance of the service to network connection loss, and the general problem of coping with event loss. In the publish-subscribe framework, event delivery is not guaranteed in the case when the intended receiver is not available, and there is no mechanism to obtain the lost events.

The P2PSIP protocol is currently in the early design stage and there is no agreement on how it would operate. However, most of the challenging research problems described in the section on collaboration middleware that concern the session management, presence and event management service would also apply to P2PSIP, since this protocol should support the full functionality of SIP over a P2P overlay network. Another challenging problem would be the transparent integration of P2PSIP with Internet-based SIP infrastructures or IMS.

The performance of P2PSIP will depend on the operation of the underlying overlay. The overlay will also support all other functions of the collaboration middleware. Therefore, a good overlay design is crucial for the support of collaboration functions. Here, the opportunity for using information from the collaboration middleware layer to optimize overlay operation becomes important, as will be shown in the next section.

5 Vertically Integrated Overlay

Structured overlays are touted to play the role of an application layer internet, potentially bridging the gap among heterogeneous networks by providing a standard API based interface for applications, and managing the underlying operations such as routing, transparently. However, in mobile (either logical or physical) environments structured overlays are harder to realize and maintain. A simple approach – called the *layered design* – is to assume that the networking layer takes care of any and all intricacies arising from mobility, and thus just deploy an overlay on top, treating the underlying layer as a black box. Such a layered design risks losing out on cross-layer optimization opportunities, which affect performance, besides potentially jeopardizing the feasibility of such a system to start with. Experiments in mobile ad-hoc environments do confirm this intuition, demonstrating that a cross-layer approach performs better, where the underlay's routing mechanisms are integrated with the overlay's routing mechanisms. A good summary comparing layered and integrated overlay designs can be found in [21]. Recent efforts have pushed this idea even further, using overlay like approach to infact perform the underlay's routing itself [10], simultaneously realizing a DHT by default.

In this part of the chapter we will first review some recent *standalone* ideas, and then propose a speculative mechanism of how these apparently different ideas

together potentially can help realizing a structured overlay in a highly dynamic environment as is witnessed in mobile environments. The standalone ideas have been tested in isolation. While the proposed speculative mechanism is one of the best bets given current understanding of overlay networks, it nevertheless currently lies in the realms of future work.

We next provide a summarized survey of these stand alone mechanisms, followed by our vision of how several of these can be combined together to provide the basic routing and indexing infrastructure for the upper layer middleware. We also will point out how information from the upper layer can be exploited to make the underlying overlay more robust and reliable.

5.1 In the Realm of Ringless Routing

Structured overlays, e.g., Distributed Hash Tables (DHTs) provide essential indexing and resource discovering in distributed information systems. Typically, structured overlays are based on enhanced rings, meshes, hypercubes, etc., leveraging on the topological properties of such geometric structures. The ring topology is arguably the simplest and most popular structure used in various overlays. In a ring based overlay network (like Chord [35]), nodes are assigned to distinct points over a circular key-space, and the ring invariant is said to be hold if each node correctly knows the currently online node which succeeds it (and the one which precedes it) in the ring. The ring is both a blessing and a curse. On the one hand, an intact ring is sufficient to guarantee correct routing. Hence, historically, all existing structured overlays have de facto considered it necessary.

Traditionally DHTs implicitly assume that any pair of nodes can interact directly with each other. Under such an assumption, the ring is relatively straightforward to realize. However, in some networking environments, particularly mobile, wireless and ad-hoc, assuming that a network layer routing mechanism provides such pairwise connectivity and then realizing the structured overlay on top of the network is at least an inefficient approach since it ignores cross-layer optimization opportunities. Moreover, even on the internet infrastructure and over even a moderately small and dedicated infrastructure like PlanetLab researchers have observed non-transitive connections [14], which violates the all-pair connectivity assumption. To make things worse, large scale networks are prone to failure of individual participants, and in peer-to-peer networks, peers often go offline and come back online, leading to churn in the network. Each and all of these factors make it harder to maintain the ring invariant, becoming a performance bottleneck in the best case, and jeopardizing the functionality altogether in the worst. There are of-course many self-stabilization mechanisms which try to ensure ring integrity, with varying degree of success, but these do not fundamentally change the fact that traditional DHT routing strategies require the ring integrity as a necessary condition, even while actually it is just a sufficient condition. Next, we describe some recent works which try a different approach, investigating whether the routing strategy can be changed

so that the DHT works even when the ring invariant *is not* or *can not* be met. In some sense, these approaches complement self-stabilization mechanisms, reducing the DHTs reliance on the ring-invariant on the one end, even while of-course the performance of the system improves if and whenever the ring invariant is actually met.

5.2 Virtual Ring Routing

Virtual ring routing (VRR)[10] is a DHT style "overlay" layer approach used to define the underlying network's routing mechanism. It is implemented directly on top of the link layer and provides both traditional point-to-point network routing and DHT routing to the node responsible for a hash table key, without either flooding the network or using location dependent addresses. While traditional DHTs take for granted point-to-point communication between any pair of participating nodes, VRR extends the idea, using only link layer connectivity. Essentially this means that the VRR scheme relaxes the traditional DHT assumption of a completely connected underlying graph.

Each node in VRR has an unique address and location independent fixed identifier, organized in a virtual ring, emulating Chord style network. Each node also keeps a list of r/2 closest clockwise and counter-clockwise neighbors for the node on the virtual ring. Such a set of neighbours is called the node's virtual neighbour set (*vset*). Typically, members in a node's *vset* won't be directly accessible to it through the link layer. Thus each node also maintains a second set called the physical neighbour set (*pset*), comprising nodes physically reachable to it through the link layer. Quality of the link is taken into consideration in choosing members of *pset*, and is updated regularly with changing conditions.

VRR sets up and proactively maintains routing paths called *vset paths* to each member in its *vset*, and these paths are typically multi-hop using *pset* nodes. This provides a routing table at each node, comprising of routes to its own *vset* nodes, as well as *vset paths* that pass through it. This is enough to support point-to-point communication between any two nodes in the system, since one can follow the *vset* neighbors to reach the destination along the ring. In practice, VRR performs significantly better, because a node can exploit the routing entries from other *vset paths* that pass through it. Figure 8 shows an example virtual ring, and illustrates how VRR works.

While proposed originally for network routing, where *pset* is determined by link layer connectivity, the idea of a VRR is potentially applicable even for other kinds of connections, including of-course hybrid physical networks, but also abstract ones like social network graphs. For routing and indexing services for a collaboration middleware, using such a social network to determine *pset* is particularly alluring, given that social network links can inherently provide security and robustness, as will be briefly explained later when we describe how ideas from *VRR*, *Fuzzynet* and

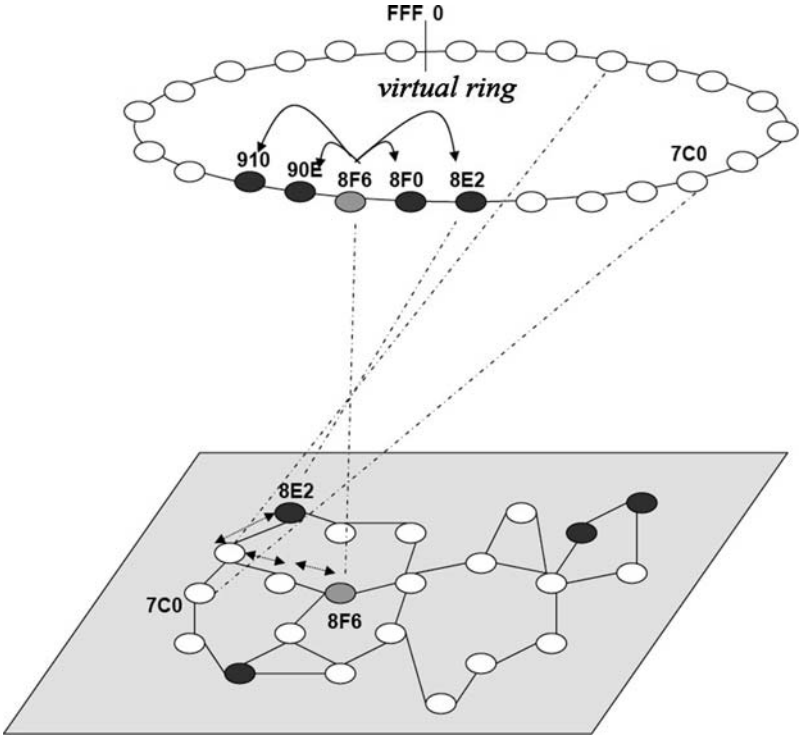


Fig. 8 The actual pair-wise connectivity of peers is used to establish route to the virtual ring predecessors and successors. For example, 8F6 has neighbors 8F0, 8E2, 90E and 910 in the virtual ring, though it is has actual connections with some nodes which are located in other arbitrary points on the virtual ring. As per the original VRR paper link state connections are used to determine pair-wise connectivity, however any other kind of abstract connections like social acquaintance may also be used instead. Now if 8FC wants to communicate with 7C0, it will choose to forward the messages greedily towards 8E2, however on the way, this message will go through a node which has direct connection to 7C0, and hence the message will be forwarded to its destination. This figure is reproduction the virtual ring example provided in the original paper (Caesar et al., 2006)

social networks can all be put together to realize a new vertically integrated overlay which we tentatively call *SocialCircle*.

5.2.1 Fuzzynet

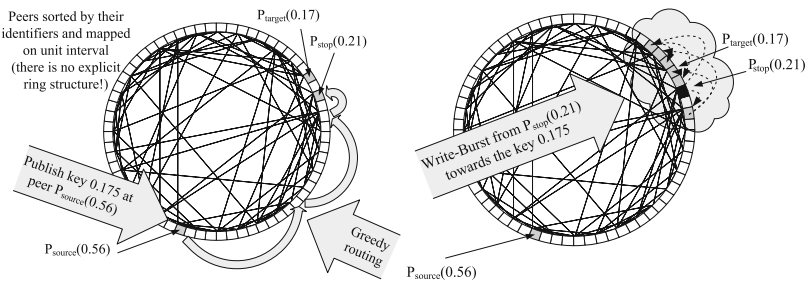
Traditional DHT designs rely on the existence of a ring in order to deterministically guarantee that any node can be reached from any other node by traversing the ring sequentially. While the ring ensures connectivity among any pair of nodes, ensuring the ring itself is challenging and expensive. In a system with large number of nodes and under continuous churn, guaranteeing the ring invariance continuously is unrealistic. Thus, even though it is true that if the ring exists then it guarantees routing,

in practice since the ring invariance itself is often not met, routing failures are also relatively frequent. Fuzzynet [16] is a recent approach which uses a probabilistic approach for both routing and data placement in the DHT, but manages to successfully perform DHT operations even when the ring invariant is not met.

The essential idea in *Fuzzynet* is as follows. As in ring based DHTs, Fuzzynet assigns nodes unique identifiers on a ring. However Fuzzynet drops the requirement for every peer having a predefined deterministic responsibility range on the identifier space. Instead, it uses a probabilistic responsibility "fuzzy" approach, where a data item will most likely be stored on a peer whose key in the identifier space is close to the hash value of that data item (data key). Secondly, each peers need to know some other peers in its neighborhood in the ring. This is however different from knowing the precise successors or predecessors on the ring, but just some nodes in the vicinity, and hence there is no need to actively maintain this information. Data is replicated in Fuzzynet, by gossiping the data replicas in the vicinity of the data position on the identifier space. Such dissemination of data replicas does not significantly increase the overheads of Fuzzynet, since all the realistic systems (which use the ring structure) employ replication for fault tolerance and persistence anyway. An useful consequence of such data replication in Fuzzynet is the fact that a simple greedy routing query will find one of the replicas w.h.p. given sufficient network connectivity, even if the ring invariance is violated.

5.2.2 Lookup (Read) Operations on Fuzzynet

Lookup routing employs a greedy routing algorithm, where messages are forwarded everytime minimizing the distance to the target. The routing terminates if a data item D, which was looked-up is found. However, since there are no ring links and no predefined responsibility ranges, a peer might end up in a situation where it does not have any links which would lead the query closer to the target, nor does the node hold the requested data locally. In such a case the lookup query terminates



(a) Greedy-Approach (b) Write-Burst

Fig. 9 Schematic example of the two phases for publish (write) in Fuzzynet: (a) Greedy-Approach to reach in the vicinity where the data should be stored and (b) Write-Burst to disseminate the data in that region in a fuzzy manner

unsuccessfully. Nevertheless, analysis proves and experiments demonstrate and validate that with realistic parameters (e.g., in the networks with ***O(log N)*** degree and typical peer replication cost), data is found w.h.p. if it was published before.

5.2.3 Publish (Write) Operations on Fuzzynet

The high guarantees for Fuzzynet lookups lie in the exploitation of a particular Small-World property, namely the clusterization property, during the data writing phase. The write operation is performed in two stages and stores data *D* on *r* peers (replicas) in the vicinity of the data key *D(key)*. The first stage is similar to the lookup (read) phase and uses greedy routing to find one of the peers which are close enough to the data key *D(key)*. Once the write operation reaches the vicinity of the targeted key location, the data is seeded in the nearby peers by the self-avoiding multicast (a controlled "Write-Burst". The underlying idea is to use the clusterization property of the network and to reach as many peers as possible in the *D(key)* vicinity. The multicast has two parameters – f_n *fanout* and *depth*. A peer contacts its fn closest neighbors to *D(key)* and requests to store the data item *D* as well as to continue the multicast process with reduced *depth*. The multicast avoids the peers which have been visited (already store data *D*) and terminates when depth reaches zero. Data *D* is seeded (stored) on all the peers reached by the multicast-burst. The two phases of the write process are shown on Fig. 9.

Experimental results show Fuzzynet to provide routing success which is upto an order of magnitude better than ring based routing (as in Chord). Fuzzynet's ability to realize DHT's indexing service and data retrieval without requiring the ring invariant makes it particularly suitable for systems not only with high churn, but also potentially in systems incurring mobility. Realizing the Fuzzynet rather than ring based DHTs on top of a social network is again relatively simpler since social network links need not be sufficient to guarantee connections to establish and maintain a DHT ring. Thus, we speculate that a hybrid approach using Fuzzynet and VRR can potentially be a good underlay for supporting collaboration middlewares, since they can explicitly take into account the nuances of social networks (implicitly present in collaboration activities) and mobility. This motivates us to explore a new DHT style overlay network which we will call *Social Circle*, the design of which and associated research issues will be elaborated at the end of this chapter.

Before doing so, we will first take a brief detour to explore some other identity related issues relevant in realizing a decentralized collaboration middleware.

5.3 Identity Crisis in a Large-Scale Decentralized World

In large-scale decentralized systems, there are two distinct identity related issues that may arise. Firstly, because of node mobility and dynamic IP address assignments, nodes' physical/IP address change over time. Many peer-to-peer networks

and applications do not care about the specific peers, as long as they are connected to some peers. However other applications can benefit or even need to relocate an existing peer. For example, in a peer-to-peer storage system, if a peer goes offline temporarily and returns back, it is useful to be able to access the content stored in it, allowing for more efficient storage system redundancy maintenance algorithms. In other situations, being able to locate back a specific peer may be even more crucial, because of shared past interactions, and the trust and social relations established among the peers. This is particularly important in supporting a collaboration middleware. A simple solution to this issue can be to use a logically centralized directory service storing the up-to-date peer-to-address mappings. However, given that many of the peer-to-peer networks themselves work as decentralized directory services, one can also imagine using the peer-to-peer network itself as a self-referential self-contained directory service to store meta-information about the participants, including their current physical address. This basic idea has been proposed independently (and varying in details) in several academic as well as deployed peer-to-peer networks, including P-Grid [4], Microsoft's Peer Name Resolution Protocol (PNRP) and Skype. A similar self-referential directory can of-course in turn also be used to support *presence*, particularly in conjunction with a publish-subscribe mechanism!

Another aspect of identity in decentralized settings is that users can create bogus identifiers. A major security threat in such systems is if a resource rich adversary creates many bogus identifiers (Sybil attack [13]), then it can disrupt the functionalities of the system (denial-of-service), as well as more actively hurt the other genuine users. While this is a serious concern in general, in a social network context (which is inherently present – explicitly or implicitly – among collaborating individuals), this is easier to thwart, because social links are established based on either pre-existing real life interactions or referrals, or even if the link is established virtually, still it is sustained and rated based on real interactions among the players. Thus, social links can be a good mechanism to thwart multiple bogus identifiers. Notice that a limited number of identifiers for the same individual is not a threat of the same scale as a Sybil attack. Also, it really does not matter in many situations as to who is the real person behind the virtual persona, but what matters more is how this persona behaves. Trust and reputation based on the social network can thus further limit the effect of spurious as well as misbehaving peers. For example, SybilGuard [38] is one recently proposed approach which uses the social network to limit the effects of Sybil attacks in a decentralized system.

5.4 DHT Based on a SocialCircle

The social network can be useful both to improve the performance of normal routing, and to provide new functionality. Imagine that a user wishes to find a new team member with certain desired knowledge or experience. Ordinarily, the social network would be used in such a case as a medium for a broadcast query. In this work, we show how this function can be achieved much more efficiently.

Previous attempts have used social network links to bolster DHTs [26], prefering social links whenever possible, but nevertheless using the "normal" random DHT links otherwise. Such an approach still relies on using the "untrusted" links most of the time, but was perhaps as good as it gets under the older regime of DHT designs, where a completely connected underlying graph and ring invariance were considered necessary. Since then, *VRR* and *Fuzzynet* have respectively demonstrated overlay designs which work even otherwise. A hybrid of these two overlays has the potential to be a suitable substrate for collaboration middlewares in a decentralized setting. Another attempt to use social networks for routing in mobile ad-hoc networks [12] used centrally located "bridge nodes" that exchanged information about their egocentric social networks. However, in this approach, routing was still done using a network-layer ad-hoc routing protocol that just benefited from the information from the social network. In our approach, the social network itself is the basis of routing decisions, eliminating the need to exchange summary information. We would like to highlight that unlike the previous portion which was a summary of current state of the art, the following material in this chapter is currently speculative, and comprises mainly of open research issues.

5.4.1 VRR Adaptation for SocialCircle

Besides the *pset* and *vset* in *VRR*, we need another metalayer in SocialCircle, which comprises of all the social network links of a node, and call this the node's *sset*. In a wired network, one can assume that the *sset* of SocialCircle is used the same way as the *pset* in original VRR proposal. In a mobile or heterogeneous environment, whenever *sset* nodes can not be reached efficiently and directly using network level routing, the *pset* can be used for multi-hop routing in a manner similar to the original VRR. The *sset* will define a multi-hop route to the next *vset* node. Corresponding to each such point-to-point routes to *vset* members, a routing table entry will be maintained at each node. And similar to VRR, SocialCircle will also use the routing entries of other routing paths that go through a node to avoid navigating the virtual ring sequentially.

SocialCircle is developed as part of the mTeam project[3]. We present results of preliminary evaluation of the proposed system. The purpose of our experiment was to examine the VRR functioning in social networks. We have made tests on real social network graphs and randomly generated social network graphs. We have measured average stretch, which is an average ratio between the number of hops traversed by a message from a source to a destination and the length of the shortest path between this source and destination in the social network [10]. Stretch is a measure of how well the overlay network uses the underlying social network topology. A broadcast query would be wasteful in terms of overall messages used, but could discover a shortest path between the querying node and the intended receiver. SocialCircle routing is much more efficient, but can increase the length of the end-to-end path.

In the experiment we run VRR on a social graph. When the network gains a stable state, each node starts sending messages to 3 random nodes. This scenario assures that every node will be used and also ensures that even distant nodes will be on end-to-end paths, which are usually longer than average network shortest paths. The test results show the impact of increasing the number of nodes on average stretch and a comparison of stretch for shortest paths of different length. Each test was performed 3 times and average results were calculated. Random social networks were generated with the same parameters, using the "Scale Free" algorithm in the Pajek software (Pajek).

5.4.2 Average Stretch of SocialCircle

Figure 10 shows the average stretch for different network sizes. Stretch in SocialCircle on random social networks increases with the number of nodes. However, it does not exceed 1.7 up to 200 nodes, and stays below 1.5 for network sizes up to 170 nodes.

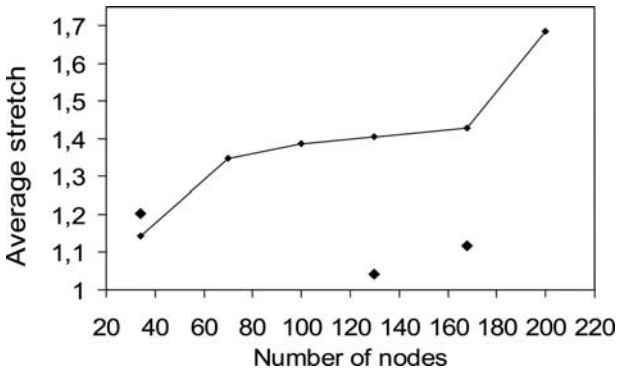


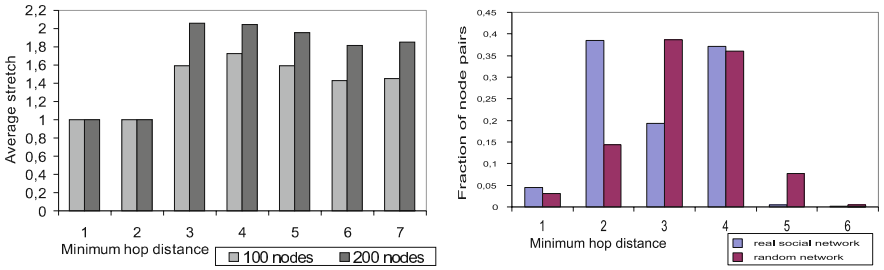
Fig. 10 Average stretch. Unconnected points show results for real social networks

Points connected with a line are test results on random social networks. The three remaining points that are not connected with lines are results of tests on real social networks. Results are quite similar at a small number of nodes (test on a sample network of 34 nodes). For real social networks with a higher number of nodes, stretch is much smaller than for random social networks.

To conclude, SocialCircle works well overall, with stretch below 70%; the approach works even better on real social network graphs, where stretch has been calculated as: 21% for a 34 nodes network, 5% for 130 nodes and 12% for 168 nodes. The reasons for these differences are discussed below.

5.4.3 Average Stretch Distribution

Figure 11a presents a distribution of average stretch for shortest paths of different length, for two different network sizes: 100 and 200 nodes.



(a) average stretch (b) minimum distances

Fig. 11 Results of experiments. (a) presents average stretch per minimum distance. (b) presents distribution of minimum distances on real and random graphs

In both examples, there stretch is equal to 1 for shortest paths of one or two hops. These results prove that SocialCircle does not make mistakes for reaching close nodes. Stretch begins to increase at 3 hops. In the 100 nodes network, stretch is about 1.6, whereas for the 200 nodes network it is almost 2.1. For the 100 nodes network, the highest stretch is at 4 hops. In the 200 nodes network, stretch is highest at 3 and 4 hops.

For longer shortest paths, stretch is decreasing, and the differences between both networks are constant. These results are quite similar to results obtained for VRR [10].

5.4.4 Distribution of Minimum Distances

Figure 11b presents a distribution of minimum distances for a real and random social network. In the real network, most paths are up to 4 hops distance, and shortest paths of 2, 3 and 4 hops make up 94% of all paths. In the random social network, the distribution of paths is shifted towards higher distances, mainly on 3 and 4 hops. However, there are also relatively many shortest paths of 5 hops. This difference, together with the results presented on Figure 9, explains why SocialCircle performs better for real social networks. Since paths in real social networks are shorter, and stretch of SocialCircle for shortest paths of 1 and 2 hops is equal to 1, the results of SocialCircle routing on real social networks will be much better than for random social networks.

The presented preliminary results of the evaluation of SocialCircle indicate that for real social networks, the stretch of SocialCircle is likely to remain very low. This means that SocialCircle is able to route queries quickly, which would be of a

special significance for session establishment and event notification in collaborative applications.

5.5 The Outlook for Future Works

5.5.1 Fuzzynet Adaptation of SocialCircle

The *SocialCircle* can be made further robust by reusing the ideas from *Fuzzynet*. Specifically, the *vset* need not comprise of precise predecessor/successors on the virtual ring, but instead use just a random set of neighbors from the vicinity and a fuzzy data placement based on gossiping over the *vset*. Such extensions are part of our future work, and has not been done yet.

5.6 Good and Better Peers

Using cooperative applications in a heterogeneous environment of mobile and fixed devices imposes a natural requirement for the diversification of peers. In fact, even P2PSIP foresees that possibility, as some SIP user agents may connect to P2PSIP peers using the P2PSIP client protocol. This solution is highly similar to a superpeer architecture. Some mobile peer may become superpeers just because they have Internet connectivity and are willing to share it with others. On the other hand, these peers may not be capable of running all superpeer functions that could be handled by a fixed node. For that reason, a three-tier hierarchy is a natural choice for the stratification of the peers according to their capabilities.

On the other hand, the question remains "how should different kinds of peers communicate?". A typical solution is to let superpeers communicate using a structured overlay, and peers communicate only with superpeers. However, such an architecture may be unsuitable in a mobile environment, where it is easy for a peer to lose contact with his superpeer. A higher connectivity among ordinary peers may be more desirable. In a different solution, all peers would use SocialCircle for communication without superpeer services, or for superpeer discovery (at the first or second tier). Third-tier superpeers receive service requests from their children in the superpeer hierarchy and serve this requests using SocialCircle routing.

An issue that remains is to determine which peers are capable to serve as third-tier superpeers. Based on information from the presence or trust management service, the overlay can automatically choose best candidates for superpeers or for peers needed for a special function (such as trusted routing). Note that this choice is not simple, since the overlay contains information about the availability and trust of individual peers, but does not contain an ordering of all peers. Obtaining all values and ordering them is not scalable. To solve this problem, it is possible to apply distributed sorting based on gossiping in the SocialCircle [22].

Another method of selecting superpeers would be to choose peers from the centre of community clusters in the social network. This method would have the advantage of supporting efficient message delivery for services such as publish-subscribe, as the superpeer could be used to receive publications and issue notifications to all peers in the cluster. The clustering would ensure that few messages would have to be sent to distant peers. To create community clusters and select appropriate superpeers, distributed clustering (community detection) protocols based on gossiping may be applicable [37].

6 Conclusions

Collaboration is one of the most promising applications of mobile network technology. The very nature of collaborative applications makes them suitable for a Peer-to-peer model, since many of these applications rely on sharing of information (expert knowledge, work results, new ideas) by people. Such sharing is inherently peer-to-peer.

Yet, in order to enable sharing, collaborative applications must have a number of functions that may require support of an infrastructure. The functional requirements for most collaborative applications have been described in this chapter. The required functions can be partially implemented using the Session Initiation Protocol and its P2P extension, P2PSIP. The emerging standard of P2PSIP points out a direction of enabling collaboration in mobile P2P environments. However, the full support of collaborative P2P applications requires further research work in the area of overlay design. In this chapter, we have explored a particular dimension of overlay design: the use of additional social network information from the collaborative middleware layer by a vertically integrated P2P overlay.

We outline how several currently stand-alone innovations can be put together to realize a peer-to-peer overlay for mobile environments, and also identify the outstanding research challenges to achieve the same. A common theme of the discussed innovations is the use of information from the collaboration middleware for the optimization of overlay operation. We have presented *SocialCircle*, with early results of a speculative approach that integrates these research ideas by using only the social network directly to create a structured overlay, which is incidentally also a first of its kind in terms of realizing the idea, even though many researchers in the community have speculated the need and possibility of such an approach [38], and devised partial solutions [26]. Preliminary results indicate that SocialCircle is likely to route messages very efficiently on real social networks, enabling efficient session establishment and event notification. These results prove the optimization potential of a vertical integration. Research that will further validate SocialCircle is already underway, together with several projects worldwide that attempt to exploit the powerful trends that bring ubiquitous, nomadic collaborative applications almost within our reach.

Acknowledgements

Most of the SharedMind implementation was carried out by Sally Ang and Michał Karwowski. The work on FuzzyNet [16] was carried out in collaboration with Sarunas Girdzijauskas, Wojciech Galuba, Vasilios Darlagiannis and Karl Aberer.

References

1. Groove networks. http://www.groove.net/ (2004)
2. Freemind. http://freemind.sourceforge.net/ (2008)
3. mteam: A creative environment for mobile knowledge workers. Research project supported by the Polish-Singaporean research program, http://mTeam.pjwstk.edu.pl/ (2008)
4. Aberer, K., Datta, A., Hauswirth, M.: Efficient, self-contained handling of identity in peer-to-peer systems. IEEE Transactions on Knowledge and Data Engineering **16**(7), 858–869 (2004). DOI 10.1109/TKDE.2004.1318567
5. Auvinen, A., Vapa, M., Weber, M., Kotilainen, N., Vuori, J.: Chedar: Peer-to-peer middleware. In: Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS) (2006)
6. Baset, S., Schulzrinne, H., Matuszewski, M.: Peer-to-peer protocol (p2pp). Internet Draft, draft-baset-p2psip-p2pp-01 (work in progress) (2007)
7. Bisignano, M., Di Modica, G., Tomarchio, O.: Jmobipeer: a middleware for mobile peer-to-peer computing in manets. In: Proc. of the 25th IEEE Int'l Conf. on Distributed Computing Systems Workshops (ICDCSW 2005), vol. 791 (2005)
8. Bryan, D., Jennings, C.: A p2p approach to sip registration and resource location. draft-bryan-sipping-p2p-01 (work in progress) (2005)
9. Bryan, D., Lowekamp, B., Jennings, C.: Sosimple: A serverless, standards-based, p2p sip communication system. In: Proceedings of the 2005 International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA 2005) (2005)
10. Caesar, M., Castro, M., Nightingale, E., O'Shea, G., Rowstron, A.: Virtual ring routing: network routing inspired by dhts. In: SIGCOMM, Proceedings, vol. 36, pp. 351–362. ACM New York, NY, USA (2006)
11. Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., Gurle, D.: Session initiation protocol (sip) extension for instant messaging. IETF RFC 3428 (2002)
12. Daly, E., Haahr, M.: Social network analysis for routing in disconnected delay-tolerant manets. In: MobiHoc, Proceedings, pp. 32–40. ACM Press New York, NY, USA (2007)
13. Douceur, J.: The sybil attack. In: Peer-To-Peer Systems: First International Workshop, IPTPS, Revised Papers. Springer (2002)
14. Freedman, M., Lakshminarayanan, K., Rhea, S., Stoica, I.: Non-transitive connectivity and dhts. In: WORLDS, Proceedings (2005)
15. Garcia-Martin, M., Matuszewski, M., Beijar, N., Lehtinen, J.: Sharing files with the session initiation protocol (sip). Internet Draft, draft-garciasipping-file-sharing-framework-00 (work in progress) (2007)
16. Girdzijauskas, S., Galuba, W., Darlagiannis, V., Datta, A., Aberer, K.: Fuzzynet: Zero-maintenance ringless overlay. Tech. Rep. LSIR-REPORT-2008-006, EPFL (2008)
17. Greene, S.: Characteristics of applications that support creativity. Communications of the ACM **45**(10), 100–104 (2002)
18. Handley, M., Jacobson, V., Perkins, C.: Sdp: Session description protocol. IETF RFC 2327 (1998)

19. Herbjornsen, S.: Software support for creativity. Tech. Rep. TDT 4735, System Engineering, Department of Computer and Information Science, Norwegian University of Science and Technology (2003)
20. Horozov, T., Grama, A., Vasudevan, V., Landis, S.: Moby-a mobile peer-to-peer service and data network. In: International Conference on Parallel Processing, Proceedings, pp. 437–444 (2002)
21. Hu, Y., Das, S., Pucha, H.: Peer-to-peer overlay abstractions in manets. In: J. Wu (ed.) Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks, pp. 845–864. Auerbach Publications (2005)
22. Jelasity, M., Kermarrec, A.M.: Ordered slicing of very large-scale overlay networks. In: Proc. Sixth IEEE International Conference on Peer-to-Peer Computing P2P 2006, pp. 117–124 (2006). DOI 10.1109/P2P.2006.25
23. Kawulok, L., Zielinski, K., Jaeschke, M.: Trusted group membership service for jxme (jxta4j2me). In: Proc. IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob'2005), vol. 4, pp. 116–121 (2005). DOI 10.1109/WIMOB.2005.1512958
24. Kortuem, G.: Proem: a middleware platform for mobile peer-to-peer computing. ACM SIGMOBILE Mobile Computing and Communications Review **6**(4), 62–64 (2002)
25. Maibaum, N., Mundt, T.: Jxta: a technology facilitating mobile peer-to-peer networks. In: Proc. International Mobility and Wireless Access Workshop MobiWac 2002, pp. 7–13 (2002). DOI 10.1109/MOBWAC.2002.1166946
26. Marti, S., Ganesan, P., Garcia-Molina, H.: Dht routing using social links. In: The 3rd International Workshop on Peer-to-Peer Systems. Springer (2004)
27. Roach, A.: Session initiation protocol (sip) – specific event notification. IETF RFC 3265 (2002)
28. Rosenberg, J.: The extensible markup language (xml) configuration access protocol (xcap). IETF RFC 4825 (2007)
29. Rosenberg, J.: Simple made simple: An overview of the ietf specifications for instant messaging and presence using the session initiation protocol (sip). IETF draft draft-ietf-simple-simple-02 (work in progress) (2008)
30. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: Sip: session initiation protocol. IETF RFC 3261 (2002)
31. Schulzrinne, H., Gurbani, V., Kyzivat, P.: Rpid: Rich presence extensions to the presence information data format (pidf). IETF RFC 4480 (2006)
32. Schwotzer, T., Geihs, K.: Shark-a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups. Journal of Universal Computer Science **8**(6), 644–651 (2002)
33. Shneiderman, B.: Creativity support tools. Communications of the ACM **45**(10), 116–120 (2002)
34. Singh, K., Schulzrinne, H.: Peer-to-peer internet telephony using sip. In: NOSSDAV, Proceedings, pp. 63–68. ACM New York, NY, USA (2005)
35. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. Networking, IEEE/ACM Transactions on **11**(1), 17–32 (2003)
36. Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., Peterson, J.: Presence information data format (pidf). IETF RFC 3863 (2004)
37. Yoneki, E., Hui, P., Chan, S., Crowcroft, J.: A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In: MSWiM, Proceedings, pp. 225–234. ACM New York, NY, USA (2007)
38. Yu, H., Kaminsky, M., Gibbons, P., Flaxman, A.: Sybilguard: defending against sybil attacks via social networks. In: SIGCOMM, Proceedings, pp. 267–278. ACM New York, NY, USA (2006)