On P2P Collaboration Infrastructures

Manfred Hauswirth, Ivana Podnar
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Stefan Decker
Digital Enterprise Research Institute
National University of Ireland
University Road, Galway, Ireland

Abstract

BSCW [8] and Groove [10] have become the two de-facto standards for collaboration over the Internet. They offer a wealth of functionalities, support a variety of possible collaboration styles, and are applied frequently in research and business projects. While being useful and powerful for standard scenarios, they suffer from some problematic aspects if support for distributed ad-hoc and mobile collaboration is needed. Both are centralized systems which require the setup and maintenance of a server. Setting up such an infrastructure for short-term and ad-hoc collaboration of mobile users is not feasible since it requires the infrastructure to be in place and configured a-priori, which is costly in terms of hardware, software, and time. Thus they are not very adequate for flexible, short term collaborations, a genuine building block of many light-weight distributed mobile collaboration scenarios. In this paper we discuss how P2P approaches could be applied to remedy these shortcomings and to what extent existing technology can already be applied "out of the box."

1. Introduction

Key characteristics of distributed mobile collaboration are the requirements for flexible interaction styles among users, and ubiquitous mobile access to resources and collaborators. Server-based applications can be applied here and work very well in practice. However, they do this at a certain cost in the literal sense. Setting up a BSCW [8] or a Groove [10] server is expensive in terms of time, costs for software and hardware, and especially because continuous administration and maintenance is necessary. In many settings, for example, at project meetings, trade fairs, or conferences, this is to much of an overhead, and a more flexible support for short-term and ad-hoc collaboration is required that takes into account user mobility. Users want to collaborate at a meeting but, on the other hand, may want to keep up the collaborations at their home base while having the data they carry with them accessible to entitled collaborators. These requirements call for a more flexible support for collaboration which additionally should be able to rely on less infrastructure which is potentially heterogeneous. This may seem as a contradiction, but in fact in such mobile collaboration scenarios users usually require less sophisticated support as long as their basic needs are satisfied. The existing server-based solutions are often too heavy in terms of functionalities as most of the time mobile users only require archival storage and sharing of documents, search for documents and users, and basic communication (active and reactive). Additionally, the issue of possibly limited functionality of mobile devices has to be taken into account which prevents the use of sophisticated platform services beyond these basic needs.

In this paper we argue that collaboration needs to get more flexible, make use of existing resources provided by the participants, should be easier to set up, and support virtual organizations and teams collaborating for limited time spans. As this implies that teams can rely on pre-existing infrastructure only to a limited extent, we believe that P2P systems have a lot to offer in this context as they specifically target the use of resources "at the edge" of the network, i.e., resources provided by the users of a system themselves, and require minimal additional infrastructure. On the other hand, they may introduce new problems that do not exist in the server-based solutions since they are not as mature and a number of problems in P2P still awaits being solved.

In terms of supported functionality we take a bottom up approach and try to address the basic collaboration needs first: distributed storage, distributed search to discover information, self-organized setup of the infrastructure, management of simple (un-)structured data with known semantics, efficient and flexible communication taking into account user mobility, and efficient dissemination of notifications according to current user needs.

A few other projects have defined similar requirements and try to address them. Croquet [14] is an open-source system which aims to support large-scale collaboration and resource sharing among large numbers of users. Croquet users have the ability to create and modify a personal information space and create fully dynamic connections to any other Croquet spaces and network-deliverable information resources (such as the WWW). Croquet is also supposed to support transactions, provide real-time synchronization and communication, sophisticated GUIs, etc. Thus it requires quite elaborate infrastructures, though they claim to follow a P2P approach. In contrast to Croquet our goal is to be light-weight with minimal resource requirements to be able to provide P2P collaboration support within reasonably short time and add new functionalities incrementally based on user feedback. In this paper we sketch an architecture for P2P collaboration in heterogeneous environments. A related architecture is defined in [7] where the authors define a P2P-based architecture for distributed and mobile collaborations, but in contrast to ours, target ad-hoc usage scenarios.

2. Scenarios and Requirements

Collaboration scenarios have recently been enriched by the proliferation of mobile devices with high-bandwidth network connectivity. Worker and team mobility has reached a level at which flexible and simple setup of provisional teams is a necessity. People who participate in such teams need instant and ubiquitous access to resources and services regardless of their current location. Furthermore, they need effective means to communicate and cooperate with other team members. It must be noted that teams are today increasingly dynamic and subject to frequent changes.

Let us consider three different collaboration scenarios covering different time spans: ad-hoc, short-term, and long-term collaborations. *Ad-hoc collaborations* offer temporary support possibly in a limited geographical area, and enable group members to flexibly interact and communicate with limited collaboration functionalities. Typical examples are meetings, conferences, or conventions where people meet, exchange contacts, ideas, and knowledge, but do not work together on a common product or to satisfy a mutual goal. *Short-term collaborations* cover limited time spans, and enable virtual organizations and teams to collaborate in order to satisfy a mutual objective, e.g., create a project proposal, jointly work on a paper, or integrate software components. Such collaborations require knowledge and data exchange, and are based on a trust relationship between team members. Therefore, we assume they can access and utilize common resources. *Long-term collaborations* are set up by collaborating organizations and teams for longer time periods, e.g., the duration of a project. As this type involves a longer time period and the involved people and utilized resources fluctuate less, it is necessary to offer flexible and versatile interaction and cooperation capabilities and services. Usually a centralized infrastructure is used for this type.

A single person may be involved in a set of different collaboration scenarios at the same time. First of all, the person must be able to find and access both required resources and available services. Resources are, for example, his/her personal calendar, files stored at the corporate server, or common team data, while a service can be conference call support, notification of document changes or collective editing. Efficient distributed search based on semantic descriptions of the involved data, users, services, etc. are therefore essential for systems supporting mobile collaborators, as discovery of any type of resources is probably the most basic service. Moreover, distributed storage, and secure and authenticated access to resources is needed. Efficient, scalable distributed search (using simple predicates) is already supported by P2P systems [1, 4, 13, 15], while semantic (structured) search and higher-level search predicates are being researched at the moment. Also large-scale distributed storage ("making the Internet your hard-disk") is still an open problem under active research [12]. Yet exploring whether P2P systems can meet the requirements here is important because if the on-going efforts are successful we are one step closer to systems that require less or even no a priori infrastructure.

Next, the person must be able to flexibly interact with other team members. Therefore, since teams may be subject to frequent changes, the system needs a team membership management service which should be augmented by a trust model. Security issues in distributed systems are largely understood and standard solutions exits. However, with highly distributed systems such as P2P systems or social networks the issue of trust gets central importance as the quality of interaction depends on the level of trust a party can put into another. In ad-hoc and short-lived collaborations a person meets unknown people whose reputation cannot be assessed a priori. Even with perfect security in place, this cannot be addressed adequately as security (authenticity, confidentiality, non-repudiation, data integrity, standard access control, etc.) has no means to assess whether a party behaves according to the agreed standards. Put simple, security lacks the "social" aspect, i.e., trust. Thus, a distributed trust-model and reputation management framework is required that does not rely on a centralized infrastructure as, for example, eBay [2].

Furthermore, expressive group communication models based on the publish/subscribe interaction style are required for efficient delivery of notifications to mobile team members [11]. Publish/subscribe enables team members to issue a subscription, i.e., a "permanent query"/filter, and the infrastructure delivers the information to the member at the time the requested information becomes available. For example, a team leader may be interested in the editing status of a document and wants to be notified when the document status changes. Publish/subscribe is essential for disseminating alert messages, or urgent notifications requiring immediate action. Assuming a mobile environment where a single user may have many different access points, publish/subscribe must rely on the information provided by a presence service which keeps track of user status and access point [6].

Keeping in mind different collaboration scenarios, we can examine different technological strategies for providing support for collaboration: a standard centralized solution, a pure P2P approach, and a mixed model comprising peers that can rely on some pre-existing infrastructure.

The *centralized solution* is expensive as it requires continuous maintenance and relies on centralized collaboration servers. Typical examples are BSCW and Groove. Taking into account different time spans and variance in collaboration teams, it seems mainly adequate for long-term collaborations.

A *pure P2P approach* uses a P2P overlay network without need for extra infrastructure or setup since team members provide the computers and software, i.e., the peers, that build up the infrastructure. However, the existing solutions currently only provide efficient search, but are still not mature enough to enable efficient distributed storage, higher-level and structured search predicates. Many system already support simple collaboration among users, typically via a "chat" metaphor. This approach would be suitable for ad-hoc and short-term collaborations, but presently can be considered for supporting ad-hoc scenarios which do not require high-reliability and availability of resources.

The *mixed model* seems a feasible intermediate solution between the centralized and pure P2P approach. It uses a reliable super-peer network for storage while guaranteeing data availability (nevertheless, some of the resources may still only be available if a user is online). This means that some infrastructure has to be in place and available at all times, which also creates setup and maintenance costs. The main question here is who is providing this infrastructure, and what are the advantages in comparison to the classical server-based approach. An obvious advantage would be that P2P systems, and therefore also the super-peer network forming the system backbone, are designed to be self-organizing, and require minimum administration. It remains to be seen whether such systems can surface, and provide support for short-term and long-term collaborations.

3. Architecture

Taking into account the pros and cons of the available technological solutions, it seems feasible and economically justified to employ P2P technology for building collaboration systems, especially for ad-hoc and short-term scenarios. Therefore, we design an architecture for a P2P-based collaboration system taking into account the requirements given in Section 2. We assume that a system uses a P2P overlay network, either a pure P2P solution, or a super-peer enhanced P2P overlay. Then a peer in the collaboration system would have the architecture shown in Figure 1.

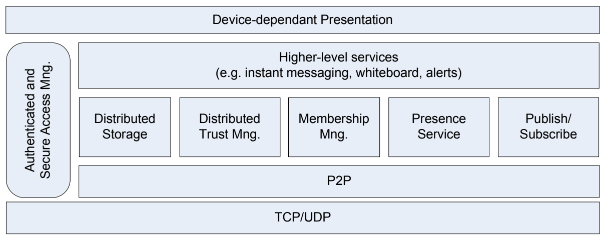
Device-dependant Presentation
Authenticated and Secure Access Mng.
Higher-level services (e.g. instant messaging, whiteboard, alerts)
Distributed Storage
Distributed Trust Mng.
Membership Mng.
Presence Service
Publish/ Subscribe
P2P
TCP/UDP

Figure 1. Architecture for P2P collaboration systems

The architecture is layered comprising transport layer, peer-to-peer overlay, lower-level services, and higher-level services with an add-on for device-dependent presentation. The transport layer provides the basic network communication between peers. The peer-to-peer overlay on top of this layer supports basic (self-organization) of nodes, indexing of of data (documents, users, system information, etc.), and efficient distributed search, either as a pure P2P solution, or involves a super-peer network to increase resource availability. This provides the basic substrate for the five basic lower-level collaboration services which offer supporting services for building end services for mobile collaborators. We have identified five basic services: distributed storage, membership management, distributed trust management, publish/subscribe, and presence service.

Having these lower-level distributed services in place, the job of building and deploying various collaboration tools becomes quite straightforward. Therefore, we assume that a number of services tailored for different user groups will be provided as higher-level services, e.g., document versioning, instant messaging, alerts, or common whiteboard. For example, instant messaging comes at nearly no cost on top of a P2P system.

The last architectural layer has the role of adjusting the presentation of higher-level services to different terminals taking into account user preferences. Like in any distributed system authenticated and secure access management is essential for collaboration systems, and it relates to P2P overlay, lower-level and higher-level services.

Let us briefly discuss lower-level services and their relationship to P2P overlay since these represent the most challenging part of the architecture. The main issues to be solved by a distributed storage system are data availability, data consistency, data confidentiality and resilience against attacks. Data availability means that data always needs to be available despite offline peers. Thus usually data is replicated or some form of erasure coding[1] is used. Replication

[1]Data is segmented and encoded such that if m out of n parts can be

always comes at the cost of having to provide functionality that ensures consistency among the replicas. Since data is stored on untrusted peers it must be encrypted to ensure confidentiality. Moreover, a solution which addresses attacks on the data in the presence of colluding peers has to be included. As all of these problems are of significant complexity in a large-scale distributed system, no standard system exists so far.

Trust and reputation management are at the focus of current research in distributed systems. The research community has identified this recently as one of the basic building blocks to enable large-scale distributed systems such as P2P-based systems, service-oriented computing, and mobile ad-hoc networks. The scale, distribution, and characteristics of these systems dictate that also trust and reputation needs to be managed in a distributed fashion to avoid single points of failures. A number of approaches which integrated well with the P2P paradigm have been proposed already, for example, [2], but the area is still subject to ongoing research. Membership management on the other hand comes at very low cost. Group management is a basic problem P2P systems need to tackle anyway to be able to provide their service. Putting higher-level semantics on top a P2P membership service is thus quite straight-forward.

The publish/subscribe (P/S) service enables asynchronous and flexible group communication. It allows the system to inform interested and available users about changes in system, availability of new data, etc. To some extent it can be seen complimentary to the search functionality provided by the P2P layer. With the functionalities of the P2P layer the user can actively send out queries while the P/S service allows the user to express interests reactively, i.e., "permanent queries." The presence service keeps track of the applied terminals and user status taking into account user-defined restrictions regarding visibility to others.

In the following we will take a more detailed look on distributed storage (Section 4) and the P/S and presence services (Section 5) as they seem to be the hardest ones to accomplish in a P2P architecture.

4. Distributed Search and Storage

In this section we take a closer look at current, advanced P2P technologies to support distributed search and storage. Distributed search is a prerequisite for distributed storage as it offers the functionality which is usually provided by a directory in a standard file system. In the simplest case search is needed to discover a resource. If a resource is large, then it may have been split into smaller segments which again would be stored on arbitrary peers. Thus again distributed search is needed to recover all parts to reconstruct the original resource. Also when replication comes into play, distributed search is relevant. It is necessary to discover replicas and provide support to keep them up-to-date. On the other hand the search mechanism is influenced by the replication mechanism as the consistency guarantees provided by the replication mechanism, determine the freshness of information the search mechanism can return.

Thus having a distributed indexing system at hand enables the discovery and access to any resource in a distributed system. Additionally these systems typically offer replication and load-balancing for the index information, i.e., the location of any resource will be found at any time, but it may not be able to access it if the corresponding peer is offline. To increase availability of the data, everyone who has downloaded a resource, of course could again enter it in the index. This would already provide a very simple notion of distributed storage, without much consistency guarantees though.

To clarify functionalities and terminology, it is also important to stop viewing current P2P systems just as distributed database systems. If they were, distributed storage would already exist. Someday P2P systems may reach this stage, but at the moment they are quite a bit away. Typical P2P systems, such as Napster, Kazaa, eDonkey, Gnutella [4], or distributed hash tables, such as P-Grid [1], Pastry [13], or Chord[15], are *location systems*. They basically provide a distributed index that can be maintained in the presence of changing peer populations, node failures, and network separations. To ensure fault-tolerance they use replication which is applied to the index, not to the indexed data itself. Replication of data in large-scale distributed systems always suffers from the problem that it simply takes to long to copy/move data among peers.

However, replication is required to ensure data availability. Simple replication strategies on a per-file basis scale only for small file sizes up to maybe 100kB. With larger sizes and just a bit of dynamicity in the system (joining/leaving peers, mobility, etc.) the system would break down immediately. The distributed index can be used to track replicas, but the actual data transfer defeats efficiency. Just think of the situation when the system dynamically decides to replicate a single 10MB file. Then an "unfortunate" person having the data to be replicated on his/her disk (because the system decided to have a replica on this computer at some earlier time), would suffer as the typical 100-200KB ADSL uplink would be quite overloaded for some time. As P2P systems are supposed to be very dynamic, this would happen very frequently and this strategy is not possible.

To remedy this situation, distributed archival storage projects such as OceanStore [12] thus split resources into "digestible" pieces and distribute them among the peers.

retrieved the original data can be reconstructed.

Coding strategies are needed to detect changed pieces, and the content must be encrypted which comes at the cost that standard data manipulation operations become very expensive. OceanStore actually tries to solve these problems. However, it is still a long way to go until an efficient implementation becomes available.

In terms of data manipulation operations, at the moment P2P systems work very well for read-only or rather static data sets. When is comes to updates, P-Grid is one of the very few systems (at least the only distributed hash table) that supports this [5]. However, updates in P2P systems usually mean index updates. In principle, data updates would be possible based on this functionality by sending notifications to data replicas. Also in Freenet[3] and Gnutella data updates would be possible in principle because these systems have intrinsic data replication, which is nearly identical to the "you download, you become a replica" model. However, consistency of updates has not been extensively studied for these systems, so they are theoretically not modeled and understood.

The key question for the distributed storage in a P2P-based collaboration system thus is, which data should be replicated. Just replicating all data typically defeats any distributed scheme since it is of no use to replicate data that nobody will ever access. Additionally, it is the question whether complete consistency is required all the time or some relaxed model would be tolerable ("better old data than no data"). We claim that in many scenarios a relaxed consistency model would suffice. We also claim that many successful systems do on-demand replication/caching, tolerate lack of information, and work with not up-to-data data. Just take Google for an example, where the average age of the index data is one month as it takes that long to crawl the Web. Still, it is very useful. Loosening consistency requirements will make the system design a bit different. However, we think that it would allow us to devise more pragmatic and simpler designs that will work in practice. In practical systems it is often the case that people would say consistency is absolutely necessary. Then if they think it over for some time and check the concrete impacts, they usually find ways to successfully live with a certain amount of inconsistency.

For a fully decentralized, distributed system, a possible solution with relaxed consistency could look as follows:

- When a peer downloads data, it becomes a replica, i.e., it adds the new location to the index. All data contains a unique ID, so all data can be discovered easily via the P2P system. The secure distributed generation of unique IDs is solved (for example in P-Grid). The index additionally stores a version vector with the locations (who has which version of the data).
- In case of an update, all replicas are being informed. If a replica wants to update, it may look for the highest version via the index and download from any source. Peers which are not online will learn about the update when they get online again. The consistency guarantees for such push/pull schemes have already been analyzed in P2P setting, for example, in [5]. The concrete policy is subject to user configurations (when, how often, if at all, only if the user agrees, etc.). The user may decide to live with a not up-to-date copy or simply invalidate the local copy.
- Queries always include the version of the found data. The user may decide what to use. Before using the data, the user can always perform a quick search and check whether he/she would access the most recent data (this is cheap in P2P systems).
- Conflicting updates require manual resolution. In fact there are no good automatic schemes that always work as this essentially boils down to the distributed cache reconciliation problem. Already distributed file systems like the Andrew file system (AFS) and Coda (its successor) had this problem 10-15 years ago and still no universal solution is in sight.
- If data is unavailable (all replicas are offline), the requester can subscribe to the system to be notified when the data becomes available again (as we have a P/S service in the architecture anyway).

Basically the same statements and approaches can be applied for super-peer-based solutions. The advantages of a super-peer solution are smaller scale and better availability characteristics of super-peers. Yet the same problems need to be addressed as in the case of a pure P2P approach, and, additionally, it has to be guaranteed that a super-peer infrastructure that can be trusted exists.

5. Publish/Subscribe and Presence

Publish/subscribe (P/S) systems offer support for collaboration scenarios that deal with $n : m$ communication with information coming from multiple and heterogeneous sources and targeting numerous users. P/S provides means to define expressive subscriptions describing information and data properties the subscriber is interested in. Subscriptions are used as data filters, because they are matched to data coming from various sources (publishers) prior to delivery to interested destinations (subscribers). A P/S system provides an efficient service which pushes the data at the time of its publication to interested subscribers. Communicating parties, publishers and subscribers, interact asynchronously by generating and consuming short information items, i.e., notifications. We assume the usage of content-based and, in particular, context-sensitive subscriptions.

Since the research in the area of P/S systems has so far mainly focused on optimizations assuming static environments, a number of open problems and challenges remain to be addressed to provide efficient solutions for P2P collaboration systems. The major challenge for P/S in such environments is related to the design of efficient routing strategies that can deal with network changes ("churn") while preserving high-expressiveness of subscriptions and low latency for delivered data. P/S has often been promoted as adequate for highly dynamic mobile environments since it enables loosely-coupled communication adaptable to a changing number publishers and subscribers. Nevertheless, research in this area has just recently recognized open issues introduced by mobile environments [9]. Furthermore, most systems assume a fixed reliable network of servers as a backbone, connecting publishers to subscribers which we actually want to avoid as much as possible. Routing algorithms typically utilize delivery trees over this backbone with sophisticated procedures for notification filtering.

P2P systems offer interesting properties P/S systems could exploit. The goal is to provide "infrastructure-less" P/S systems on top of P2P systems, i.e., a P/S overlay on top of a P2P overlay. However, currently there are no P/S systems that utilize P2P characteristics, such as efficient routing, redundancy, and load balancing, to build flexible and efficient P/S solutions. P2P could be used to maintain a distributed index of active subscriptions, but the question remains how to match a published notification to the existing subscriptions in the index and how to build delivery trees based on P2P systems. These and related questions are subject of ongoing research.

Presence Service. Presence service maintains and offers information on users' presence and contact information, i.e., whether a user is on-/offline and how to contact the user. This includes the current user communication capabilities and preferences with respect to the applied terminal, application, and user state. Furthermore, a user can define a default communication point which is activated in case the current presence information is unavailable.

P/S is the natural interaction style for the presence service. Users providing information about their presence are presence publishers, and users interested in presence information are presence subscribers. Therefore, presence implementation can largely rely on an existing P/S implementation taking into account an extension related to user privacy. In other words, users need to be able to grant access to their presence information. Furthermore, the update and retrieval of this information must be secured and authenticated.

6. Conclusions

In this paper we have argued that light-weight infrastructures are required to support short-lived and ad-hoc collaboration in dynamic and mobile environments. We have presented a P2P-based architecture and discussed to what extent existing technologies can be exploited in the implementation. We are convinced that P2P-based collaboration is a paradigm that meets the requirements of users, but there is still a lack of enabling technologies which have to be researched and implemented.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems*, 2001.

[2] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the 10th International Conference on Information and Knowledge Management (2001 ACM CIKM)*, pages 310–317. ACM Press, 2001.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[4] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), Jun. 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[5] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[6] M. Day, J. Rosenberg, and H. Sugano. A Model for Presence and Instant Messaging, February 2000. RFC 2778. http://www.ietf.org/rfc/rfc2778.txt.

[7] S. Dustdar and H. Gall. Towards a Software Architecture for Distributed and Mobile Collaborative Systems. In *26th International Computer Software and Applications Conference (COMPSAC 2002)*, 2002.

[8] F. FIT and O. S. GmbH. BSCW (Basic Support for Cooperative Work), 2005. http://bscw.fit.fraunhofer.de/.

[9] G. Mühl, A. Ulbrich, K. Herrmann, and T. Weis. Disseminating information to mobile clients using publish/subscribe. *IEEE Internet Computing*, 8(3), May 2004.

[10] G. Neworks. Groove, 2005. http://www.groove.net/.

[11] I. Podnar and I. Lovrek. Supporting Mobility with Persistent Notifications in Publish/Subscribe Systems. In *Proceedings of the Third International Workshop on Distributed Event-Based Systems (DEBS '04)*, pages 80–85, May 2004.

[12] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free Global Data Storage. *IEEE Internet Computing*, 5(5), September/October 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 2001.

[14] D. A. Smith, A. C. Kay, A. Raab, and D. P. Reed. Croquet - A Collaboration System Architecture. In *First Conference on Creating, Connecting and Collaborating through Computing*, 2003.

[15] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, 2001.