S. Zöls, R. Schollmeier, A. Tarlano, W. Kellerer: The Hybrid Chord Protocol: A Peer-to-peer Lookup Service for Context-Aware Mobile Applications, in Proc. of IEEE 4th International Conference on Networking (ICN), Reunion Island, April 2005. LNCS 3421.

http://www.springeronline.com/sgw/cda/frontpage/0,11855,5-40109-22-45347206-0,00.html

The Hybrid Chord Protocol: A Peer-to-peer Lookup Service for Context-Aware Mobile Applications

Stefan Zöls, Rüdiger Schollmeier, Wolfgang Kellerer, Anthony Tarlano

Institute of Communication Networks, Technische Universtität München, Munich, Germany
Future Networking Lab, DoCoMo Communications Labaratories Europe, Munich, Germany
{stefan.zoels, ruediger.schollmeier}@tum.de, {kellerer, tarlano}@docomolab-euro.com

A fundamental problem in Peer-to-Peer (P2P) overlay networks is how to efficiently find a node that shares a requested object. The Chord protocol is a distributed lookup protocol addressing this problem using hash keys to identify the nodes in the network and also the shared objects. However, when a node joins or leaves the Chord ring, object references have to be rearranged in order to maintain the hash key mapping rules. This leads to a heavy traffic load, especially when nodes stay in the Chord ring only for a short time. In mobile scenarios storage capacity, transmission data rate and battery power are limited resources, so the heavy traffic load generated by the shifting of object references can lead to severe problems when using Chord in a mobile scenario. In this paper, we present the Hybrid Chord Protocol (HCP). HCP solves the problem of frequent joins and leaves of nodes. As a further improvement of an efficient search, HCP supports the grouping of shared objects in interest groups. Our concept of using information profiles to describe shared objects allows defining special interest groups (context spaces) and a shared object to be available in multiple context spaces.

1. Introduction

P2P networking refers to a class of systems, applications and architectures that employ distributed resources to perform routing and networking tasks in a decentralized and self organizing way. Within the last few years, P2P traffic has become a major part in IP networks. This lead to an increasing attention of P2P in the research community, and many new P2P protocols have evolved dealing with the core problem of P2P systems: how to find a node sharing the required object and thereby to generate as little traffic as possible.
A very promising approach to solve this problem is the concept of Distributed Hash Tables (DHT). In contrast to unstructured P2P concepts, DHTs map keys onto nodes allowing direct localization of shared objects and therefore avoid flooding, which reduces effectively the signaling overhead in the network.
One realization of DHTs is the Chord protocol [1]. It is a scalable, distributed lookup protocol that uses a hash function to determine a node's m-bit identifier from its IP address. The same hash function is used to produce an identifier for every shared object. The identifier of a shared object is called its key k. The Chord nodes are ordered in an identifier circle modulo 2^m. Every key k is assigned to the first node in

the ring whose identifier is equal to or follows *k*. This node stores references to all shared objects in the network whose key precedes the identifier of the node. Thus flooding of query messages can be avoided, as the query can be routed directly to the responsible node.
When a new node joins the Chord ring, it has to receive all object references according to the keys it is responsible for from its succeeding node. Again when the node leaves the network, all object references have to be transferred to the succeeding node. This can result in a high traffic volume, especially when there are many shared objects (and therefore object references) in the network and when nodes participate in the overlay network only for a short period of time.
In a mobile scenario, a high rate of joining and leaving nodes (= churn rate) is common, e.g. because of the high charges in a mobile network. Besides high churn rates, also the relatively low transmission data rates and the limited resources of mobile devices demand for a low signaling overhead.
In this paper we aim at a scenario of context aware applications in particular. In a mobile ubiquitous computing environment, a multitude of nodes provide information to be found and accessed by the respective application. However, the matching between applications and the relevant context (e.g. provided by sensors) is difficult (if applications and sensors are not 'hard-wired'). Therefore we use the concept of context spaces, introduced in [2], as an application support middleware to allow an application independent access to context information. Context spaces allow sharing of information among applications and context providers by creating and maintaining context spaces as virtual containers of information of particular interest.
In a mobile scenario, such context spaces middleware could be realized by a multitude of loosely coupled nodes providing information or hosting applications requesting such, which is supported by a P2P infrastructure as described above. We abandon from centralized solutions where all object references are maintained in a highly available centralized server, to avoid the following disadvantages:

- Single point of failure
- Scalability problems caused by a high amount of information that is provided and accessed
- Maintenance work and administration costs for the operator
- Responsibility of the operator for the shared information

As we have seen, the existing Chord protocol can lead to problems when used in such a mobile scenario. However, we choose Chord as base protocol for our concept to support efficient P2P search in mobile environments. Compared to other DHT-based protocols like CAN [3] or Pastry [4], Chord is rather simple and therefore easy to adapt.
In this work we propose the Hybrid Chord Protocol (HCP), a modified version of Chord that is able to deal with the problems in mobile, wireless scenarios and to support the concept of context spaces. HCP uses two types of nodes –static nodes and temporary nodes– to avoid the traffic load that is caused by shifting object references.
HCP also addresses the grouping of shared objects in context spaces. We introduce info profiles to label all provided content in the network. Every shared object is described by an info profile which contains several keywords. By means of these keywords the shared object is assigned to multiple context spaces, each containing all

available information for a given keyword. This concept allows an easy way of finding required information simply by building the intersection of all context spaces given in the query.
This paper is organized as follows. Section 2 compares HCP to related work. Section 3 gives an overview over the architecture of HCP, while section 4 presents details of the protocol. In section 5, we evaluate the improvements of HCP by comparing the traffic load in Chord and HCP that is generated by shifting object references. Finally, we summarize our contributions and outline items for future work in section 6.

2. Related work

Currently there exists a high number of DHT based protocols which establish structured P2P networks. The most prominent among them are Chord [1], CAN [3], Pastry [4] and Tapestry [5]. Since their publication, a number of modifications to improve them, e.g. to adapt them to the underlying physical topology [6], to provide anonymity and privacy in the overlay network [7] or to allow complex queries [8] have been proposed.
However, the signaling overhead especially in unstable environments is still a problem, which we try to solve in our approach. Chord e.g. needs at least $O(\log^2 n)$ messages to repair the routing tables affected by a single node arrival or departure [9], where n represents the total number of nodes in the Chord network. Furthermore, DHT-based applications often store a large number of object references that have to be transferred to other nodes upon a node arrival or departure. If we take into account the short session lengths measured in FastTrack and Gnutella networks, this may result in high maintenance traffic [10, 11]. The environment can be so dynamic that DHT applications provide little useful service other than constant maintenance operations. Thus we must state that especially in mobile scenarios the current solutions of structured P2P networks are hardly applicable. Another problem of DHT-based networks is that the distribution of keywords describing the shared objects is not uniform. This results in unfair storage consumption between the nodes.
Tang [9] *et al.* developed a new method to minimize the routing distance to one or two hops by distributing all membership and content location changes in form of node profiles to every node in the DHT network. Thus they can reduce routing failures significantly. However, this approach only works in stable environments with low churn rates [9]. The same goal –to decrease routing distances– is targeted by S-Chord [12]. The authors propose a bidirectional ring to improve the node join and leave mechanism. So they can decrease the lookup failure rate and increase the lookup efficiency up to 50%. Hyperchord [13] uses a more aggressive scheme for maintaining routing table entries. It provides in place fixing fingers as an optimization to improve also the efficiency of node joins and leaves. However, both of them do not provide any analysis about unstable environments, and therefore can, from our point of view, not solve the problem of shifting object references caused by high churn rates.
The problem of unfair distribution of object references in a DHT-based network mentioned above is addressed in [14]. The authors propose an architecture called

keyword fusion to balance unfair storage consumption among the participating peers on a Chord ring. They can thus achieve a reduction of the storage consumption of the 5% top loaded nodes by 50%, but also can not solve the problem of shifting object references, resulting in a high amount of signaling traffic.

In general, the adaptation of the overlay network to the underlying physical network is certainly a good approach to decrease delays and the overall data rate on the physical layer. Such an approach can avoid zigzag and unnecessary long routes in the physical layer [6]. However, as DHT networks are structured P2P networks, the node IDs would have to be bound to the node's location. Resulting, every change of the ID of a participating node equals to one node leave and one node join, which would cause a significant amount of signaling traffic. Thus such an approach can only be applied in a relatively static environment, where the nodes hardly move, i.e. do not change their physical location frequently.

Koorde [15, 16] is a DHT-based approach which achieves low maintenance overhead by employing de Bruijn graphs [17]. In this work, the authors propose an algorithm that can control the maintenance overhead by adjusting the average route length, i.e. the longer the route the smaller is the average maintenance traffic. However, it does not address the problem of high churn rates as they do not differentiate between static and mobile nodes.

The diminished Chord ring approach, proposed in [18], establishes a primal ring and a number of tree-like substructures. These substructures are employed to carry out maintenance and routing tasks that do not require the involvement of all nodes participating in the overlay network. They are mainly based on an adapted finger establishment scheme. In diminished Chord the fingers are not established depending on the distance in the ID space, but on the content and interest shared by each node. Thus the complexity of the whole architecture increases, but the overall maintenance and routing overhead can be reduced. However, especially the overhead caused by high churn rates can still be improved if this approach would differentiate between the nodes according to their resources and their uptime, as proposed in this work.

Assigning a special role to more stable nodes, characterized by longer uptimes, is also the main improvement to Chord suggested in [19]. The authors propose an architecture which increases the routing success probability by replicating shared content on stable nodes, according to the uptime of the nodes. Their basic motivation for this approach is their finding from Gnutella analyses that the longer a node is actively participating in the network, the higher is the probability that a node remains in the overlay network. Thus, data replicated on stable nodes is lost only with a significantly decreased probability compared to the original Chord approach. Furthermore, every node in a Kademlia network exploits the information routed through itself and can thus reduce the routing and maintenance overhead significantly, as it can reduce the number of stabilize and fix-finger calls. However, Kademlia does not address the problem of high churn rates, which we target in this work, and is therefore from our point of view better suited for stable environments.

To establish the context spaces mentioned in section 1, we employ so-called info profiles, which represent content and context information brought into the network by the participating peers. A similar concept of interlinking objects by means of profiles is also employed in [20] to establish a resource management framework for P2P

networks. In [20] resource definitions are used to describe shared resources, whereas we employ info profiles to set up and describe context spaces.

3. Architecture of HCP

Basic Concept

In the conventional Chord protocol all nodes in the network are considered to be equal. This equality between nodes does not exist in real world scenarios. For example in a fixed network, some Chord nodes could have a broadband flat rate connection to the Internet, while other nodes use only slow modem connections. In mobile scenarios, this situation is even more concrete, as the major part of the network is formed by mobile devices like mobile phones or PDAs, which have limited battery and storage capacity, limited transmission data rate and usually remain in the Chord network only for a short time.

HCP addresses these problems by the differentiation between static nodes and temporary nodes:

- Static nodes are highly-available nodes that form a quasi-permanent part of the HCP ring. They usually remain in the network for a longer period of time and have high data rate connections to other static nodes as well as a larger storage capacity. All object references in the network are stored at static nodes.
- Temporary nodes are all nodes that do not form a quasi-permanent part of the HCP network. In most cases, temporary nodes join the network only for a short period of time to find some particular pieces of information. In HCP, temporary nodes do not store object references. When a temporary node joins the network, all object references according to the keys it is responsible for remain with its closest static successor, in order to avoid the shifting of object references. If temporary nodes receive a request for an assigned key, they forward the request to their closest static successor that stores all object references for this key.

By the differentiation between static and temporary nodes, HCP ensures that only static, quasi-permanent nodes store object references. Thus nodes that join the network only for a short time do not cause traffic load due to the shifting of object references, beyond transferring their own object references to static nodes. Furthermore, temporary nodes with limited storage capacity are prevented from storing a lot of object references.

As static nodes store all object references available in the network, a high volume of data has to be transferred when a static node joins or leaves the network. However, this usually will not cause any problems as such joins or leaves are assumed to occur seldom and the data usually is transmitted over a fixed broadband network between the static nodes.

Realization of Context Spaces

In HCP, every shared object is described by an info profile. The definition of an info profile is depicted in Fig. 1. It consists of five parts: the name of the shared object, a

description (optional), at least one keyword, the IP address of the host, and a timestamp.

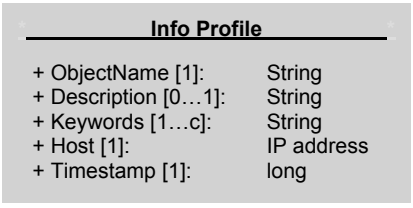

Fig. 1. Definition of an info profile.

The first string in an info profile represents the name of the shared object, while the second string gives a description about it. Then one or more keywords follow that specify all relevant context spaces for this info profile. We explain the concept of context spaces in detail in the following paragraph. Finally, the info profile contains the IP address of the host offering the object, and a timestamp to ensure that outdated info profiles can be discarded.
HCP groups all available information in context spaces. Every keyword in a shared object's info profile indicates a relevant context space for this object. As an example, Fig. 2 shows an XML version of the info profile describing the file "Beatles – A little help from my friends.mp3".

```
<InfoProfile>
   <ObjectName>Beatles – A little help from my friends.mp3</ObjectName>
   <Description>
      mp3-version of the Beatles' song "A little help from my friends"
   </Description>
   <Keyword>Beatles</Keyword>
   <Keyword>Help</Keyword>
   <Keyword>Friends</Keyword>
   <Host>123.4.5.67</Host>
   <Timestamp>0</Timestamp>
</InfoProfile>
```

Fig. 2. Example for an info profile in XML.

By means of the keywords this file is assigned to the context spaces "Beatles", "Help" and "Friends". Thus the sharing host sends this info profile to those static nodes that are responsible for the keywords "Beatles", "Help" and "Friends", i.e. are the first static successors of the hash values of these keywords. Each static node stores a list for every keyword it is responsible for. In this list all info profiles that contain that keyword are collected. These lists form the context spaces, as they hold all available information for a given keyword. In our example, the static node that is responsible for the keyword "Friends" provides all info profiles with keyword "Friends" in the according context space. Fig. 3 gives an illustration about the organization of info profiles in context spaces.

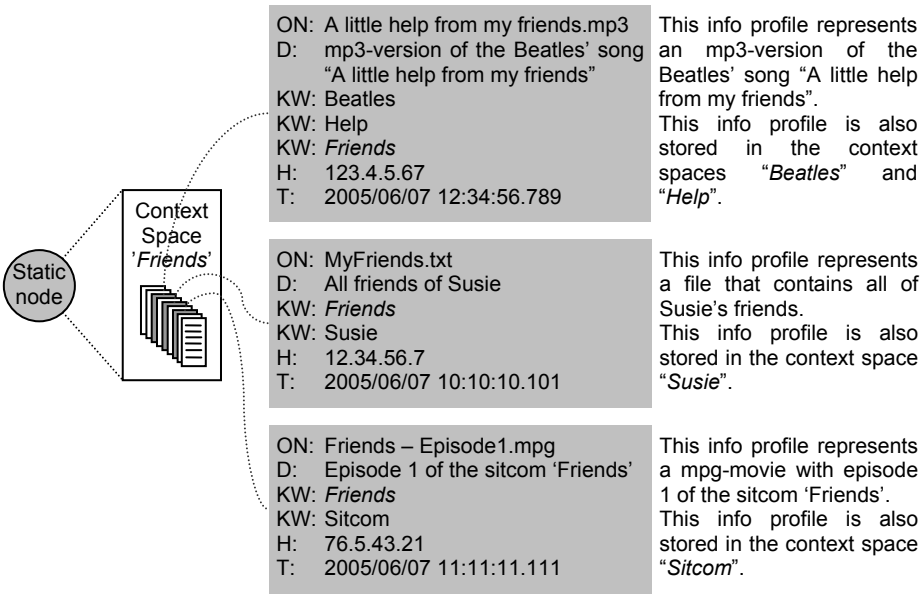

Fig. 3. Example for a context space "Friends". The responsible static node stores all info profiles with keyword "Friends" in this context space.

The timestamp in an info profile is used to recognize outdated info profiles, e.g. info profiles from nodes that left the HCP network without deregistering. When a static node receives an info profile for one of its context spaces, it sets the info profile's timestamp to the current time. After a given period of time, the info profile will be removed from the context space if it has not been renewed by the sharing host.
An application that searches the network for a particular piece of information can use the context spaces to efficiently find the requested information. It simply builds the intersection of all context spaces that are relevant for the query. Assume a query with the keywords "Beatles" and "Help". To build the intersection of these two context spaces, an application can either look for all info profiles in the context space "Beatles" that contain the keyword "Help", or it can look for all info profiles in the context space "Help" that contain the keyword "Beatles". In both cases, it will receive the same results. For this reason, it is sufficient to send the request only to one static node, which is yet another possibility to reduce signaling traffic.

4. Protocol details

In this section we describe the necessary operations when nodes join or leave the network, what has to be done when nodes want to insert information into the HCP ring, and how queries are routed in HCP.

Join
When a node, either static or temporary, joins the HCP network, it first has to determine its position on the HCP ring by hashing e.g. its IP address. Then it sets its predecessor, its successor-list and its finger table entries according to the conventional Chord algorithm. Additionally, the joining node must set a pointer-list to its *s* closest static successors. The number of entries in the static-successor-list *s* can be significantly lower than the number of entries in the conventional successor-list (which should be *O(log n)*, with *n* as the total number of nodes in the network [3]), because the session length of static nodes is assumed to be significantly higher than compared to temporary nodes. Therefore it is also sufficient to update the static-successor-list after a longer period of time than it is necessary for the conventional 'stabilize' algorithm.
New static nodes that join the HCP ring finally have to receive all info profiles they are responsible for. Therefore they send an according message to their closest static successor.

Insert
Having joined the HCP ring, a node can insert its shared objects –which are described by info profiles– into the network. For every info profile, the application hashes all keywords and searches for the responsible nodes for this info profile. Then the node contacts the ascertained nodes to ask for their first static successor and sends the info profile to these static nodes. This procedure ensures that only static nodes store info profiles and that temporary nodes have to deal only with low signaling overhead.

Query
To find a particular piece of information, the user starts a query with one or more appropriate keywords. By hashing the given keywords, the application determines the responsible nodes for the requested information.
Let us again have a look at our example, the Beatles' song "A little help from my friends", from section 3. Info profiles containing the keywords "Beatles", "Help" and "Friends", respectively, are stored at the three according static nodes. To find info profiles matching all three keywords, it is sufficient for the application to contact only one of these nodes, e.g. the static node that holds the context space "Beatles", and look there for all info profiles containing also the keywords "Help" and "Friends" (= building the intersection of the context spaces "Beatles", "Help" and "Friends").
Resulting, the querying node sends an according query message to the node that is responsible for the key (= hash value) of "Beatles", and this node –if it is a temporary node– forwards the query message to its closest static successor, which can then reply to the query. In case the responsible node is already a static node, it can immediately send a reply.

Leave
When a node leaves, it should deregister from the network, i.e. it should inform all static nodes that store info profiles of it to delete them. (In addition to that, outdated information can also be recognized by an info profile's timestamp.) Furthermore, a

leaving static node has to transfer all its stored info profiles to its closest static successor.

5. Analytical evaluation of HCP

With our analytical considerations we prove the ability of HCP to reduce significantly the signaling traffic. The traffic analyzed in this section is generated by the shifting of info profiles when nodes join or leave the overlay network. Further maintenance traffic sources are not considered, as they are the same in Chord and in HCP. In the first part of this section we calculate the average traffic load per node in the conventional Chord protocol. Afterwards, we compare the result with the traffic load that is generated in HCP.

Under the assumption that o is the total number of shared objects, and that every info profile that describes a shared object contains c keywords, $o \cdot c$ info profiles have to be stored in the whole network. This results in an average value of

$$I_n = o \cdot c / n \tag{5.1}$$

info profiles per node, where n is the number of nodes participating in the overlay network.

Each node is assumed to leave the network after an average session length T. As info profiles have to be shifted when a node joins the ring and also when it leaves again, the transfer rate per node amounts to

$$\tau_n = 2 / T \tag{5.2}$$

Assuming that b is the average size of an info profile, the average traffic load for a single node λ_n can therefore be calculated by

$$\lambda_n = I_n \cdot b \cdot \tau_n = (o \cdot c / n) \cdot b \cdot (2 / T) = (2 \cdot o \cdot c \cdot b) / (n \cdot T) \tag{5.3}$$

From this mathematics, we can conclude the advantage of HCP in comparison to the conventional Chord protocol. In HCP, the traffic load that is caused by temporary nodes is zero, as temporary nodes do not store info profiles. Additionally, static nodes –which store all info profiles– usually remain in the network for a long period of time. Due to their high session length T_{Static} the average traffic load per node can be decreased significantly, as shown below.

Assume an HCP ring with n nodes, whereof $x \cdot 100\%$ of the nodes are static and $(1-x) \cdot 100\%$ of the nodes are temporary, with $0 < x < 1$. The static nodes have an average session length which is α times the average session length of a conventional Chord node:

$$T_{Static} = \alpha \cdot T \tag{5.4}$$

The session length of the temporary nodes can be negligible, because temporary nodes do not store info profiles and therefore do not generate traffic load by shifting them.

Under these conditions, every static node stores

$$I_{Static} = (o \cdot c) / (x \cdot n) = (1 / x) \cdot I_n \quad (5.5)$$

info profiles.
These info profiles have to be shifted when a static node joins and leaves the HCP network, so the transfer rate of a static node is given by

$$\tau_{Static} = 2 / T_{Static} = 2 / (\alpha \cdot T) = (1 / \alpha) \cdot \tau_n \quad (5.6)$$

This results in an average traffic load per static node of

$$\lambda_{Static} = I_{Static} \cdot b \cdot \tau_{Static} = (1 / x) \cdot I_n \cdot b \cdot (1 / \alpha) \cdot \tau_n = (1 / (x \cdot \alpha)) \cdot \lambda_n \quad (5.7)$$

As stated above, the traffic load of temporary nodes $\lambda_{Temporary}$ is 0. Resulting, we have an average traffic load for all nodes in the HCP network of

$$\lambda_{HCP} = (x \cdot n \cdot \lambda_{Static} + (1 - x) \cdot n \cdot \lambda_{Temporary}) / n = x \cdot \lambda_{Static} = (1 / \alpha) \cdot \lambda_n \quad (5.8)$$

Thus we can state that the average traffic load per node that is generated by the shifting of info profiles in a HCP network can be decreased by a factor of $1 / \alpha$ in comparison to a conventional Chord network, with $\alpha = T_{Static} / T$.

6. Conclusion and future work

In this paper we proposed a new architecture named HCP that establishes a structured P2P network and shows a good performance especially in unstable, e.g. mobile environments where high churn rates of the participating nodes must be expected.
The architecture is based on the Chord protocol and extends it by two different node classes, namely static nodes and temporary nodes. Therefore we can reduce the overall routing and maintenance traffic of Chord significantly, but additionally keep the advantageous properties of Chord, like the guaranteed availability of shared content. The good performance of HCP, i.e. the significant reduction of traffic load that is generated by the shifting of info profiles, is proven by our analytical evaluation in section 5.
HCP also allows a very efficient implementation of the context spaces concept. Within HCP the context spaces are automatically established and maintained by assigning and transferring info profiles to the according nodes.
Currently we are working on the implementation of Chord and HCP in ns-2. Thus we want to verify our analytical results and be able to evaluate the performance of HCP in a completely simulated mobile environment. To be able to analyze the performance of HCP in a larger scale, i.e. with a higher number of nodes, we are currently also developing a simulation that scales up to a few million peers.

References

[1] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications" presented at ACM SIG-COMM Conference, 2001.

[2] A. Tarlano and W. Kellerer, "Context Spaces Architectural Framework" presented at SAINT 2004 Workshop on Ubiquitous Services, 2004.
[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network" presented at ACM SIGCOMM Conference, 2001.
[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems" presented at IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.
[5] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-Scape Overlay for Service Deployment" *IEEE Journal on Selected Areas in Communications*, vol. 22, 2004.
[6] L. Zhuang and F. Zhou, "Understanding Chord Performance" Technical Report CS268, 2003.
[7] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication" Cornell University Computing and Information Science Technical Report, TR2003-1890, 2003.
[8] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, "Complex Queries in DHT-based Peer-to-Peer Networks" presented at 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), 2002.
[9] C. Tang, G. Altekar, and S. Dwakadas, "Calot: A Constant-Diameter Low-Traffic Distributed Hash Table" in *Under submission*, 2003.
[10] S. Sen and J. Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks" presented at ACM SIGCOMM Internet Measurement Workshop, 2002.
[11] R. Schollmeier and A. Dumanois, "Peer-to-Peer Traffic Characteristics" presented at EUNICE 2003, 2003.
[12] V. A. Mesaros, B. Carton, and P. V. Roy, "S-Chord: Using Symmetry to Improve Lookup Efficiency in Chord" presented at 2003 International Conference on Parallel and Dis-tributed Processing Techniques and Applications (PDPTA'03), 2003.
[13] K. Lakshminarayanan, A. R. Rao, and S. Surana, "Hyperchord: A Peer-to-Peer data Location Architecture" UC Berkley Technical Report CS-021208, 2001.
[14] L. Liu and K. D. Ryu, "Supporting Efficient Keyword-Based File Search in Peer-to-Peer File Sharing Systems" IBM Research IBM Research Report. RC23145 (W0403-068), 2004.
[15] M. F. Kaashoek and D. R. Karger, "Koorde: A Simple Degree-Optimal Distributed Hash Table" presented at Fifteenth annual ACM-SIAM symposium on Discrete Algorithms, 2004.
[16] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems" presented at 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), 2004.
[17] N. D. Bruijn, "A Combinatorial Problem" *Koninklijke Nderlandse Akademie van Wetenschapen*, vol. 49, 1946.
[18] D. R. Karger and M. Ruhl, "Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks" presented at International Workshop on Peer-to-Peer Systems (IPTPS '04), 2004.
[19] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric" presented at International Workshop on Peer-to-Peer Systems (IPTPS'02), 2002.
[20] T. Friese, B. Freisleben, S. Rusitschka, and A. Southall, "A Framework for Resource Management in Peer-to-Peer Networks" presented at NetObjectdays 2002, 2002.