

HyperCuP – Shaping Up Peer-to-Peer Networks

Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl

Computer Science Department, Stanford University
{schloss, sintek, stefan, nejdl}@db.stanford.edu

Abstract. Peer-to-peer networks are envisioned to be deployed for a wide range of applications. However, P2P networks evolving in an unorganized manner suffer from serious scalability problems, limiting the number of nodes in the network, creating network overload and pushing search times to unacceptable limits. We address these problems by imposing a deterministic shape on P2P networks: We propose a graph topology which allows for very efficient broadcast and search, and we describe a broadcast algorithm that exploits the topology to reach all nodes in the network with the minimum amount of messages possible. We provide an efficient topology construction and maintenance algorithm which, crucial to symmetric peer-to-peer networks, does not require any central server nor super nodes in the network. Nodes can join and leave the self-organizing network at any time, and the network is resilient against failure.

1 Introduction

Peer-to-peer networks are envisioned to find a broad range of applications, moving way beyond their current application as infrastructure for file sharing and exchange such as in Napster or Morpheus [2]. However, current P2P approaches face unresolved problems. Genuine P2P networks such as Gnutella, consisting entirely of identical peers in terms of their role in the network, do not scale to a large number of nodes due to their use of inefficient search mechanisms relying on undirected broadcast. Other P2P approaches such as Napster are deprived of the fundamental advantage of P2P, namely the strict peer symmetry, by employing central servers. Super-peer networks such as Morpheus share this disadvantage.

In this paper, we focus on creating a deterministic network structure which can then be exploited to carry out efficient searching and broadcasting. We achieve this by organizing peers in a P2P network into a graph structure based on hypercubes in our system HyperCuP (Hypercube P2P). We present a distributed algorithm which is capable of maintaining the graph structure efficiently. In its current implementation, our approach is especially suitable for P2P networks in which desired information is widely distributed in the network, spread among a larger number of nodes. Instead of routing point queries, a query would be broadcasted along the topology, trying to pick up as

much information from peers as possible. Our broadcast algorithm achieves this at an optimal level. We describe the topology in Section 2, explain our algorithms in Section 3, outline extensions in Section 4, briefly discuss related work in Section 5 and conclude in Section 6.

2 A Hypercube P2P Topology

Peer-to-peer networks consist of nodes which are connected to each other. In the following, we state some requirements on a more deterministic organization of such networks.

2.1 Network Model

A graph topology in a P2P network is essentially established by peers being able to communicate with each other: Neighbors of a peer are those nodes in a network to which the peer can directly send messages. Assuming the existence of a transport network on top of which the P2P network evolves, this refers to the peers' knowledge of each other's transport network address. TCP connections among computers on the Internet are a possible manifestation of a link between peers in a P2P network. Hence shaping up P2P networks means to control the way peers connect in the network – to arrive at a topology whose properties are inherently known by all peers and can thus be used for efficient routing and search algorithms.

2.2 Aims and Requirements

The P2P network is to be symmetric: Each node in the network is to have identical capabilities and duties on the network. This excludes the existence of central servers which might be involved in organizing the network, such as in [2]. The network diameter Δ , defined as the shortest path between most distant nodes in terms of node hops, is to be of reasonable order, a crucial property for search and broadcast. Clearly, we want to beat the worst case of Δ at $O(n)$. The node degree is to be limited, limiting essentially the amount of TCP links a node has to maintain. Network traffic during search and broadcast should be distributed evenly among nodes in the network, assuming an even distribution of broadcast and search origins in the P2P network: Hotspots in the network should be avoided. At last, the topology has to provide redundancy. Node failures must not lead to the graph disconnecting or severely hampering broadcast and search properties.

2.3 Organizing Peers in a Hypercube Graph

Essentially, these requirements state that every node should be able to become the root of a tree spanning all nodes in the network. We arrive at an efficient version of this aim by organizing nodes as depicted in Figure 1a for a base $b = 2$ – which, drawn in 3D, turns out to be a hypercube topology. A complete hypercube graph consists of $N = b^{L_{\max}+1}$ nodes and is defined by the fact that all nodes have $(b - 1) \cdot (L_{\max} + 1)$ neighbors, $(b - 1)$ in each ‘dimension’ – where $L_{\max} + 1$ is essentially the number of dimensions spanned by the cube (in Figure 1, the cube has three dimensions). The network diameter is $\Delta = \log_b N$. As visible, this structure is symmetric, i.e. no node incorporates a more prominent position than others. This is crucial for load balancing in the network: Every node can become the source of a broadcast, the load will always be shared equally. The hypercube base b can be chosen to adjust the network diameter and node degree.

Note at this point that the construction algorithm that will be described in Section 3.2 works well with node numbers that are not equal to those in complete hypercubes, allowing for any number of peers in the network.

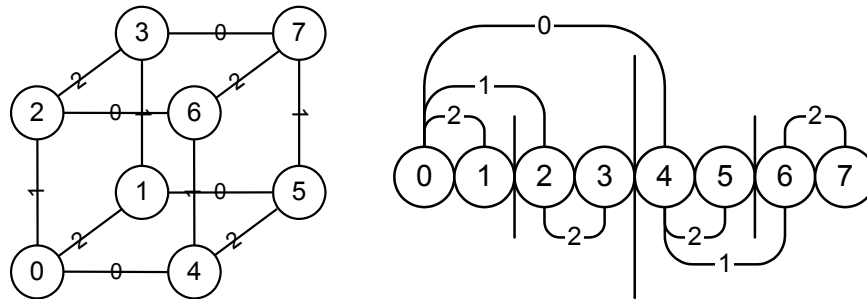


Figure 1. a. Hypercube graph b. Serialized notation (links incomplete)

To describe the topology of a graph $G = (V, E)$, we state some definitions. In the following, we will deal with hypercubes with a binary base for brevity. (Refer to [6] for an extension to bases $b > 2$.)

1. Edges in the graph are labeled: Node Y is dubbed i -neighbor of node X or $Y = iN(X)$ iff $(X, Y) \in E$ and $i = L(X, Y)$, where E is the set of edges in the graph and set L contains the edge labels. Node Y is also referred to X 's neighbor on neighbor level i .
2. Edges in the graph are undirected: $\forall i (Y = iN(X) \leftrightarrow X = iN(Y))$.
3. i -Neighbors are unique: $\forall i (Y = iN(X) \wedge Z = iN(X) \Rightarrow Y = Z)$.
4. A node can have extended neighbors $Y = N(X) = \{x_0, x_1, \dots\}(X)$, where N is termed neighbor link set, and it denotes the sequence of i -neighbors one would have to follow in the complete hypercube graph

to reach node Y from node X and vice versa (since i-neighbors are unique).

Edge labels start at $i = 0$. The maximum neighbor level of a node is termed L_{\max} . Any node X in the network maintains two sets which determine its location in the graph topology: A set of neighbor link sets $\Omega = \{\{\}, N_1, N_2, \dots, N_n\}$ and an associated set of node addresses $A = \{localhost, addr_1, addr_2, \dots, addr_n\}$. A neighbor is identified by a link set N and can be reached by sending a message to its address addr.

2.4 Broadcast and Search Algorithm

Based on this terminology, we can explain a broadcast scheme which has a great advantage: It guarantees that the set of nodes traversed strictly increases during a forwarding process, i.e. nodes receive a message exactly once. It is guaranteed that exactly $N-1$ messages are required to reach all nodes in a topology. Furthermore, the last nodes are reached after $\log_b N$ forwarding steps. Any node can be the origin of a broadcast in the network, satisfying a crucial requirement.

The algorithm works as follows: A node invoking a broadcast sends the broadcast message to all its neighbors, tagging it with the edge label on which the message was sent. Nodes receiving the message restrict the forwarding of the message to those links tagged with higher edge labels.

As an example, refer to the serialized notation of the network graph in Figure 1b (for clarity, only the links used in the example are depicted – however, one can just copy all links in Figure 1a into this notation to arrive at the full picture): Node 0 sends a broadcast – at first to all its own neighbors, viz. nodes 4, 2 and 1. Node 4 receives the message on a link tagged as a level 0 link, i.e. it forwards the message only to its 1- and 2-neighbors, namely 6 and 5. At the same time, node 2 which has received the message on a level 1 link forwards it to its 2-neighbor, node 3. In the third forwarding step, node 6 relays the message to node 7, again its 3-neighbor.

The characteristic path length [6] in this scheme can be calculated as

$$L = \sum_{i=1}^{\log_b N} \frac{(b-1)^{\log_b N - i + 1}}{(\log_b N - i)!} \cdot \prod_{j=0}^{\log_b N - i} (i + j)$$

A search in a hypercube is essentially a broadcast with a time-to-live, i.e. a broadcast with a limited scope. It too has a monotonically increasing neighbor set which means that the maximum number of nodes is reached with a given number of messages.

3 Building and Maintaining Hypercube Graphs

In the following, we will outline a distributed algorithm which allows nodes to build a hypercube topology. To maintain network symmetry, crucial for P2P networks, any node in the network should be allowed to accept and integrate new nodes into the network. Furthermore, joining and leaving the network are to consume a reasonable amount of message transmission to limit the traffic imposed on the transport network. Clearly, a joining node should not have to register with all nodes in the network, i.e. we would like the protocol to beat a message number of $O(n)$ for node joins and removals.

3.1 Definitions

To describe the maintenance algorithm, some further definitions are needed.

Link set subtraction. Since a link set N denotes a path from a node X to a node Y in the complete hypercube graph, the difference between two link sets rooted at node X and leading to nodes Y and Z , respectively, denotes the link set which describes the path from node Y to node Z . Formally,

$$N_y - N_z = (N_y / N_z) \cup (N_y / N_z)$$

Note that link set subtraction semantically equals link set addition.

Minimum link set level. The minimum link element in a link set always refers to the cluster that the link set originally intended to reach in the complete hypercube graph, before topology changes forced it to adopt additional hops. Thus,

$$\text{minlevel}(N) = \min_i (x_i \in N)$$

Graph hop distance. This metric is defined as the number and dimensionality of hops on the shortest path (or rather the link set N describing this path) between two nodes in the complete hypercube graph as

$$\|N\| = b^{\max_i (x_i \in N)} \cdot \left(\sum_{x_i \in N} b^{-i} \right)$$

Cluster and missing clusters. A cluster in the hypercube graph is formed by two nodes which are i -neighbors of each other, plus all their neighbors on higher levels. For example, in Figure 1b nodes 0, 1, 2 and 3 form a cluster on level 0, the complementary cluster of which the group of nodes 4, 5, 6 and 7 is referred to. A node has a missing cluster on level i if it does not have any node in the node's complementary cluster on level i .

3.2 Topology Construction and Maintenance Algorithm

In the following, construction and maintenance of a hypercube P2P topology shall be described. We begin by considering a network's deviation from a complete hypercube state (as stated in 3.1) first, i.e. by describing the topology's reaction to the departure of a node. The reason for describing this case first is that nodes joining just reverse the process of nodes leaving. The construction and maintenance algorithm is based on the notion that nodes in an evolving hypercube graph take over responsibility for more than one position in the hypercube. In fact, the hypercube topology of the next biggest complete hypercube graph is implicitly present in the current topology state, i.e. in the sets of all participating nodes. Upon arrival of new nodes, the complete hypercube topology will unfold again. Upon removal of nodes, other nodes jump in to cover the positions previously covered by the node that left the topology, prepared to give these positions up again as new nodes join. Since the complete hypercube topology is implicitly preserved, the broadcast and search algorithms do not have to change either.

3.2.1 Node Departure

A departing node transfers some of the connections it currently maintains to a subset of its neighbors. This subset is dubbed buffering cluster: The node's links that are to be preserved will be distributed among the nodes in the buffering cluster in a deterministic scheme. Nodes which currently maintain links to the departing node are referred to as transferring nodes: Upon construction of the distribution scheme, these nodes transfer their connections from the departing node to the buffering nodes, implementing the scheme.

Distribution scheme. In general, this scheme leads to the fact that nodes take over positions on all of their missing neighbor levels: If M is the set of a node X 's missing neighbor levels, the power set of M is the set of positions that X has to cover in addition to his own position.

Covering positions. A node is said to cover a position of another, possibly departed node, if it takes over a subset of that node's links (see below). In the more detailed description to follow, a node $V_{\text{dep}} = \{\Omega_{\text{dep}}, A_{\text{dep}}\}$ leaves the network. In Figure 2a, node 7 is about to disappear: Before it finally disconnects, it is supposed to carry out the following actions.

Determining the buffering cluster. Neighbor levels L_{min} and L_{max} span the departing cluster, i.e. the cluster which is currently represented by the departing node. Note that a node can represent a cluster that comprises more than just itself, viz. if all its highest neighbors have disappeared before and have left their positions for the currently departing node to be taken over. In the example, node 7 represents a cluster of size 1, i.e. only its own position. L_{buf} is 2: Node 7 will select node 6 to transfer its connections.

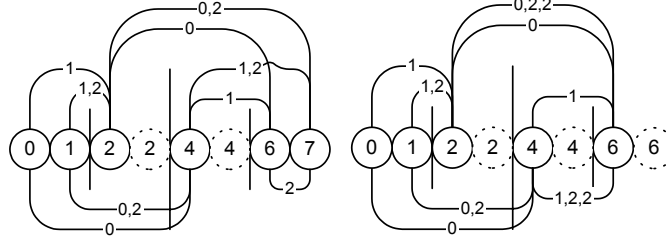


Figure 2a. P2P topology b. P2P topology after node departure

The levels are computed as

$$L_{\max} = \max_i \left(\max_j (x_j \in N_i) \mid N_i \in \Omega_{dep} \right)$$

$$L_{\min} = \max_i \left(\min_j (x_j \in N_i) \mid N_i \in \Omega_{dep} \right) + 1$$

The buffering cluster is always the cluster that is complementary to the departing cluster – hence the neighbor level on which buffering will take place is computed as $L_{buf} = L_{\min} - 1$.

Identifying actively buffering nodes in the buffering cluster. The buffering cluster may not be complete, i.e. nodes may have been departed before – hence the departing node has first to identify the nodes that still exist in the buffering cluster, all on neighbor level L_{buf} from the perspective of the departing node. These are associated with the shortest link sets in the node's set of link sets. The link sets to the still existing and therefore soon to become actively buffering nodes are assembled in Ω_{buf} . Thus, we have

$$N \in \Omega_{buf} \leftrightarrow N \in \Omega_{dep} \wedge \min_i (x_i \in N) = L_{buf}$$

In the example, node 1 alone represents the buffering cluster.

Identifying missing subclusters in the buffering cluster. Nodes take over positions on their own missing neighbor levels, and a missing subcluster in the buffering cluster is just a missing neighbor level for some nodes. Missing subclusters therefore determine the positions that nodes in the buffering cluster have taken over. For all nodes that still exist in the buffering cluster, their missing subclusters are determined: A node N_i in Ω_{buf} does not have a neighbor at neighbor level $L \geq L_{\min}$ (i.e. a missing neighbor level L) iff there is no link set N_j in Ω_{buf} for which the difference between N_i and N_j has minimum level L , as stated for any N_i and N_j in Ω_{buf} as

$$L \in M_{N_i} \leftrightarrow \neg \exists N_j, N_j \mid N_i \in \Omega_{buf} \wedge N_j \in \Omega_{buf} \wedge \min_k (x_k \in (N_i - N_j)) = L$$

M_{N_i} is the set of missing neighbor levels of the node that is referred to by link set N_i in Ω_{buf} . In the example, node 7 does not lack any subclusters.

Computing buffering node covering of the departing cluster. Each node in the buffering cluster is supposed to take over certain positions in the

departing cluster, determined by the set of its own missing neighbor levels. Since this set, M_{N_i} , is now known for all existing nodes in the buffering cluster, the positions covered by node N_i in the departing cluster can be computed as

$$\Omega_{\text{cover}}^{N_i} = N_i + P(M_{N_i})$$

This set contains a link set to each of the positions the buffering node covers in the departing cluster – adding N_i is necessary to root the link set at the position of the departing node, i.e. to ‘normalize’ it. Such a set is computed for each node N_i in Ω_{buf} , and the union of these sets contains a link set to all positions in the departing cluster, i.e. establishing a full covering:

$$\Omega_{\text{cover}} = \bigcup_{N_i \in \Omega_{\text{buf}}} \Omega_{\text{cover}}^{N_i}$$

There is now exactly one node assigned to assuming responsibility for each of the positions that will be abandoned soon by the departing node, because every link set in Ω_{cover} is associated with a node address of a node in the buffering cluster. In the example, node 6 will cover node 7’s position – indicated by the dashed depiction of node 6 at node 7’s former position, as for nodes 2 and 4 which had taken over additional positions before.

Projecting link sets back on their original positions. If the departing cluster comprises more positions than just the departing node’s position, this means that the departing node has assumed responsibility for other nodes that had disappeared before. These links will have to be transferred to the covering nodes as well – but before, they have to be traced back to the position at which they were originally rooted, before the departing node took over. A link set of such an assumed connection will contain the link set to the position that it was originally rooted at as a subset. This holds true due to the rule used for transferring connections, as stated in the next paragraph. Hence the departing node checks with all its link sets if one contains a link set N_{cover} in Ω_{cover} as a subset. (Note that this has to be carried out in an order such that link sets in Ω_{cover} that are searched for first are not themselves subsets of link sets in Ω_{cover} that are searched for later on, which is always possible.) If so, this link is destined to be transferred to the node covering the position N_{cover} , i.e. the address associated with N_{cover} in Ω_{cover} , which can be retrieved.

Transferring a projected link set. A link set N that has been successfully traced back to its original root position N_{cover} in the departing cluster will be transferred to the buffering node N_{buffer} that has been determined as assuming responsibility for it in the future. It will be transferred as

$$N_{\text{transfer}} = (N / N_{\text{cover}}) \cup (N_{\text{buffer}} - N_{\text{cover}})$$

Which essentially means that the link set’s projection back onto its original position (the set in the first pair of brackets) is then projected onto its future temporary position, the node reachable at N_{buffer} . In the example, node 7 will

transfer its connection to node 2 as new link set $\{0,2,2\}$ to node 6, as well as its connection to node 4 as new link set $\{1,2,2\}$.

Wrapping up the departure. Finally, the departing node will notify all its neighbors about the transfer of their links from itself to the buffering nodes. The contacted neighbors will then contact their associated buffering node, establish a connection with link set N_{transfer} to him and cut off their connection to the departing node. Figure 2b depicts the new topology state.

3.2.2 Link Failures

A link failure in the network leads to a node's immediate departure from the P2P topology, not being able to send any departing messages. If that happens, the topology must be able to recover and head back to a normal state. In the hypercube graph, we can always recover from a sudden node loss. The procedure is based on the axiom $iN(jN(X)) = jN(iN(X))$ (and can be found in detail in [6]): The node that is closest to the vanished node in terms of graph hop distance contacts the vanishing node's neighbors by asking its own neighbors for them. The node then carries out the node departure routine on behalf of the vanished node. This procedure does not change the message complexity as described in Section 3.3.

3.2.3 Node Arrival

To join the network, an arriving node is able to contact any of the nodes that are already connected in the hypercube graph. Node arrivals essentially follow very similar steps as node departures: The contacted node $V = \{\Omega, A\}$ selects one of its own vacant neighbor positions as integration position. It then bestows some of the connections that it currently maintains on the new node – namely exactly those it maintains due to its covering of the previously vacant neighbor position that is now to be filled with the new node.

Selecting the integration position. A node which is contacted by an arriving node will integrate the node on a vacant neighbor position with the lowest minimum level, as to balance the hypercube graph. If there is no vacant neighbor position (which is the case when all neighbor link sets that the node maintains consist of exactly one hop element), the node will be integrated on a newly to be opened highest neighbor level, $L_{\text{max}} + 1$. Link set $N_{\text{integration}}$ (which always contains exactly one element, denoting the desired integration level) resembles the integration position.

Selecting the integration control node. Since a departing node transfers its connections to exactly those nodes that are located in the cluster that is closest to it, only nodes in this particular cluster possess the information that is required to integrate a newly arriving node on that position. Therefore, the node contacted for integration first has to forward the integration request to the node that is closest to the vacant position that is to be filled. Note that

since this position used to be filled by a direct neighbor of the contacted node, the contacted node always maintains exactly one connection over which it can reach the node that is now closest to it. Out of all its link sets, the contacted node forwards the integration request to the node at link set $N_{control}$, found as

$$N_{control} = N_i \leftrightarrow N_i \in \Omega \wedge \|N_i - N_{integration}\| = \min_j (\|N_j - N_{integration}\|)$$

Forwarding the integration request. The selected integration control node, reachable at $N_{control}$, will receive the forwarded integration request with a modified integration link set on which to integrate the node, viz.

$$N^* = N_{integration} - N_{control}$$

This denotes the link set to the integration position, seen from the perspective of the integration control node.

Distributing the integration request. A departing node distributes its connections among all nodes in the cluster that it had selected as buffering cluster at the time of its departure. Therefore, all these nodes have to be notified of the fact that a node position that they possibly cover is about to be restored, i.e. acquired by the new node. The integration control node hence forwards the integration request in a local broadcast to all its neighbors in the buffering cluster – the scope of which is determined once again by levels L_{min} and L_{max} , computed in an analogous way as stated in Section 3.2.1.

Integrating the node. First of all, an integration control node establishes a connection to the new node, at link set N^* : The new node opens up a neighbor level cluster again that has previously been vacant. Then, essentially the steps undertaken in the removal of a node are reversed: Since the new node now fills a previously missing neighbor level, the integrating node can transfer a part of its maintained connections to the new node, viz. exactly those that belong to the positions that used to be covered by the integration node and will now be covered by the new node. This works in an analogous way as the operations described in Section 3.2.1. In case the integrating node does not cover a position on a particular neighbor level, the integrating node contacts its immediate neighbor on this level, calling on him to establish a link to the node. It is precisely this operation which constructs the graph from the scratch, as shown in Figure 3: In the second stage, 0 does not have any connections to transfer to arriving node 2 to provide node 2 with a required link on level 0 – hence, it calls on its own 0-neighbor 1 to cover 2.

3.3 Complexity

Assuming a relatively balanced graph structure, the algorithms as described above yield an $O(\log_b N)$ complexity in terms of messages that have to be sent in order to join or leave the network. More precisely, this holds when there are

only nodes in the graph that have $\lfloor \log_b N \rfloor - 1$ or $\lfloor \log_b N \rfloor$ non-missing neighbor levels. Note that this allows for any number of nodes in the graph.

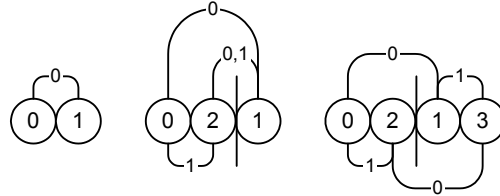


Figure 3. Example construction sequence

4 Extensions and Future Work

Balancing the graph. Assuming a relatively uniform joining and leaving pattern of nodes in the long run (‘burst’ node joins or removals are no problem), the graph will keep itself balanced since nodes actively fill vacant node positions. However, when there are nodes which in general experience a significantly larger number than joining nodes than other nodes, the hypercube graph can become unbalanced, leading to some nodes evolving to super nodes which maintain a larger number of connections and sharing higher network load. While this can be used for engineering purposes as shown in [10], it is desired to be avoided in a network with symmetric peers. Sending joining nodes on random walks through the graph before connecting them is an approach that we are currently investigating.

Ontology-based clustering. Instead of organizing the entire network in a giant hypercube, we can cluster together nodes that share interest in a particular topic and organize only them in smaller hypercubes, creating a number of hypercubes in the network. We can use the level clusters that are present in the hypercube topology anyway to map them onto ontology concepts. This method will be addressed in a future paper.

Real-time protocol. Allowing for simultaneous node joins and removals requires the handling of race conditions etc. We have a stand-alone simulation platform for our system. To arrive at this extension, we are currently implementing our algorithms on JXTA, an open-source P2P system [11].

5 Related Work

Making P2P networks scalable has recently received quite some attention. In [9], network structure and content distribution in the network are ignored, instead search algorithms are proposed as alternatives to pure broadcast. In [1], the network topology is ignored, but an attempt is made to map the

content distribution among the peers in the network. Both techniques will work fine on our topology. Distributed hash table approaches [7] such as CAN [5] and Chord [6] aim at enforcing a deterministic content distribution instead which can be used for routing point queries. While similar in terms of message complexity for joining and departing nodes, our approach specifically performs well at optimizing the network load in broadcast and multipoint search, without requiring hash functions.

6 Conclusion

We have presented an algorithm that is capable of organizing peers in a peer-to-peer network in a deterministic manner that provides efficient broadcast and search properties, including no message overhead at all during broadcast, logarithmic network diameter and resiliency towards node failure. Super peers or central servers are not required. Organizing peers in this manner is especially useful in a domain where data on a specific topic or query is to be collected from numerous peers in a P2P network. For example, the approach is well-suited to work in the domain of Semantic Web services [3]: In trying to resolve a query for a particular action at an optimized cost, offers from service providers have to be retrieved and compared. Peers representing these providers organize in a P2P network and may be queried efficiently by way of our topology and broadcasting scheme.

7 References

- [1] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.
- [2] Morpheus website. www.musiccity.com
- [3] D. Martin et al. DAML-S: Semantic Markup for Web Services. White paper, 2001, www.daml.org/services/daml-s.
- [4] G. Pandurangan, P. Raghavan, E. Upfal. Building low diameter P2P networks. In *Proc. of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, 2001.
- [5] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, August 2001.
- [6] M. Schlosser, M. Sintek, S. Decker, W. Nejdl. Shaping up peer-to-peer networks. *Technical Report*, Stanford University, April 2002.
- [7] S. Ratnasamy, S. Shenker, I. Stoica. Routing Algorithms for DHTs: Some Open Questions. In *Proc. of 1st International Workshop on Peer-to-Peer Systems*, March 2002.
- [8] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan. Chord: A scalable P2P lookup service for internet applications. In *Proc. of ACM SIGCOMM*, August 2001.
- [9] B. Yang and H. Garcia-Molina. Improving efficiency of peer-to-peer search. In *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.
- [10] B. Yang and H. Garcia-Molina. Designing a super-peer network. *Technical Report*, Stanford University, February 2002.
- [11] Project JXTA: An open, innovative collaboration. White paper, available at www.jxta.org.