HyperCuP – Hypercubes, Ontologies and Efficient Search on P2P 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 number of messages possible. We provide an efficient topology construction and maintenance algorithm which, crucial to symmetric peer-to-peer networks, does neither require a 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. Moreover, we show how our scheme can be made even more efficient by using a global ontology to determine the organization of peers in the graph topology, allowing for efficient concept-based search.

1 Introduction

Peer-to-peer networks are envisioned to find a broad range of applications, moving way beyond their current applications as infrastructure for file sharing and exchange such as in Napster or Morpheus [3]. For example, P2P networks for Semantic Web Services can be used to provide distributed access to these services without requiring a central service directory [14]. [8] uses a global service ontology which enables efficient service discovery and composition of web services. However, a fundamental building block of large-scale P2P networks is still missing: scalability. In almost all P2P networks requests for services use flooding algorithms which are based on inefficient broadcast mechanisms. We achieve efficiency by organizing peers in a P2P network into a graph structure based on hypercubes in our system HyperCuP (Hypercube P2P) which we describe in this paper. Section 2 describes the graph topology and its suitability for efficient broadcast and search. Section 3 presents a distributed algorithm which is capable of maintaining the graph structure efficiently, and elaborates the algorithm on a detailed example. In Section 4,

*On leave from University of Hannover, Germany.

we further extend this infrastructure by sketching the use of ontologies for partitioning the network.

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 [5]. 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, in 3D, turns out to be a hypercube topology with b nodes in each dimension, a topology that has been studied before in the area of multiprocessor machines [2], but under very different assumptions. 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, and L_{max} is 2). 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 3.1 works well with node numbers that are not equal to those in complete hypercubes, allowing for any number of peers in the network.

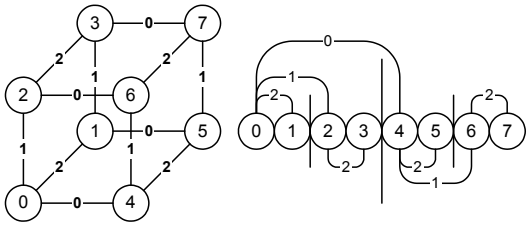

Figure 1. a. Hypercube graph b. Serialized notation (links incomplete)

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 [9] for an extension to bases b > 2.) Edges in the graph are labeled: Node Y is dubbed i-neighbor of node X or $Y = iN(X)$ iff node Y is X's neighbor in dimension i. For example, in Figure 1, node 5 is the 2-neighbor of node 4. Node 5 is also dubbed 4's neighbor on neighbor level 2. Edges in the graph are undirected, i.e. node 4 is also 5's 2-neighbor. A node can have extended neighbors Y = N(X) = $\{x_0, x_1, ...\}(X)$, 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. In our example, the neighbor link set {0,1} leads from node 1 to node 7 and back, i.e. 1 = {0,1}(7) and 7 = {0,1}(1). Since all links in the graph are undirected, the order in which the steps on the path that is described by a link set are carried out is not important, i.e. 1 = {0,1}7 and 1 = {0,1}7.

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 $\Omega = \{\{\}, N_1, N_2, ..., N_n\}$ and an associated set of node addresses

$A = \{localhost, addr_1, addr_2, ..., addr_n\}$. 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 define a broadcast scheme which 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 [9] in this scheme can be calculated as

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

which is about $0.5 \cdot \log_b N$.

A search in a hypercube is essentially a broadcast with a time-to-live, i.e. a broadcast with a limited scope. It also 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 outline a distributed algorithm which allows nodes to build a hypercube topology. Here, the major challenges in P2P networks are as follows: 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 Topology Construction and Maintenance Algorithm

In the following, we will describe our construction and maintenance of a hypercube P2P topology. The formal description of the algorithm and a proof of its completeness can be found in [9]. We will walk through an example by having 9 peers joining a network, and one peer leaving during the process, to elaborate the basic idea of the construction and maintenance algorithm.

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. The idea is to have the hypercube topology of the next biggest complete hypercube graph already 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 unfolds as needed. 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 – still, every peer receives a broadcast message exactly once.

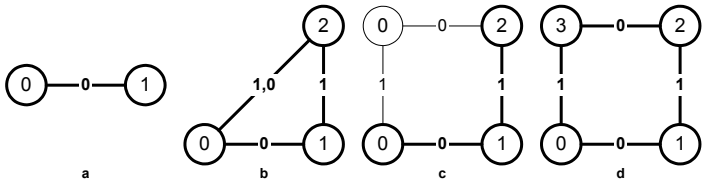

Figure 2. Network topology construction

Start. At the beginning, only peer 0 is active.

Step a. Peer 0 is contacted by node 1 which wants to join the P2P network. Peer 0 integrates peer 1 as 0-neighbor since it does not currently have any other neighbor: The peers establish a link between each other which is tagged with the neighbor set {0}. Generally, a peer integrates a joining peer on its first vacant neighbor level, the neighbor levels are ordered such that lower neighbor levels always come first.

Step b. Peer 2 contacts one of the two peers (here, we assume that it contacts peer 1) to join the network. The first vacant neighbor level of peer 1 is 1 since it already maintains a 0-neighbor, peer 0. Essentially, peer 1 opens up a new dimension for the hypercube, as depicted in Figure 2b. Peer 1 becomes the so-called integration control node for the complete integration of

peer 2 into the network: It is responsible for providing peer 2 with all necessary links – at the end of the integration process, peer 2 has to have neighbor links connecting it all currently existing neighbor levels, in order to be able to carry out complete broadcasts. Since peer 1 currently has two neighbors, a 0- and a 1-neighbor, it knows that it has to provide peer 2 with a 0- and a 1-neighbor, too. Peer 1 itself has become peer 2's 1-neighbor. Since there is currently no alternative, peer 1 selects peer 0 as the new 0-neighbor for peer 2.

However, peer 0 can only become a temporary 0-neighbor for peer 2 because it already has another 0-neighbor, namely peer 1 – and we said before that a peer can only have one neighbor per neighbor level. Essentially, peer 0 now covers a vacant position in the hypercube, i.e., it acts as if it occupies two positions in the hypercube, as depicted by the thin copy of peer 0 in Figure 2c. To mark the link between peers 2 and 0 as temporary relationship, it is tagged with the link set {0,1} (instead of {0}): This link set denotes the path from peer 2 via the position at which the link set is originally aiming to peer 0, the peer which currently covers this position. (This path is also well visible in Figure 2c.) Temporary link sets are always constructed by this rule.

Step c. Peer 3 wants to join the network. We compare three cases, viz. peer 3 contacting peer 0, 1 or 2 to join the network.

In case peer 3 contacts peer 0 to join, peer 0 follows the general rule to integrate the peer on its first vacant neighbor level – which is 1, since peer 0 has a 0-neighbor, but no 1-neighbor. As its new 1-neighbor, peer 3 will now cover the temporary position that peer 0 used to maintain in the hypercube: Hence peer 0 can pass on links that are associated with this position to peer 3. Due to the construction rule of edge labels for temporary link sets, peer 0 is able to determine that link {0,1} to peer 2 is a link that is to be passed on to peer 3. Peer 3 then establishes a link tagged by link set {0} to peer 3, as depicted in Figure 2d.

In case peer 3 contacts peer 2 to join, peer 2 decides to integrate peer 3 as its new (and non-temporary) 0-neighbor. However, it does not carry out the integration itself: Since peer 0 currently covers the position that will soon be occupied by peer 3, the integration control responsibility has to be forwarded to peer 0. Peer 2 can do so via peer 0. Note that it is always possible for peers in the network to reach the node to which they have to forward the control integration, if necessary, in one hop. We prove this in [9]. Peer 0 carries out the integration just as described above, arriving at Figure 2d.

In case peer 3 contacts peer 1, peer 1 will integrate peer 3 on neighbor level 2, i.e., it opens up a new dimension for the hypercube. This leads to a momentary misbalance in the hypercube with some peers maintaining more links than others. However, the hypercube will only become misbalanced in the long run if there are 'joining hotspots' in the network. Burst joins of peers are no problem, the structure will balance itself again in the long run.

Moreover, the information on vacant position in the structure is always spread in the network, i.e., it is likely that a joining peer will contact a network peer that has a vacant position to fill, inherently balancing the graph. In extreme cases, we have to carry out active balancing (for example, by sending joining nodes on a random walk through the network). We defer this work to a future paper. Having compared these three joining scenarios in this step, we will explore a specific joining scheme for brevity in the next steps.

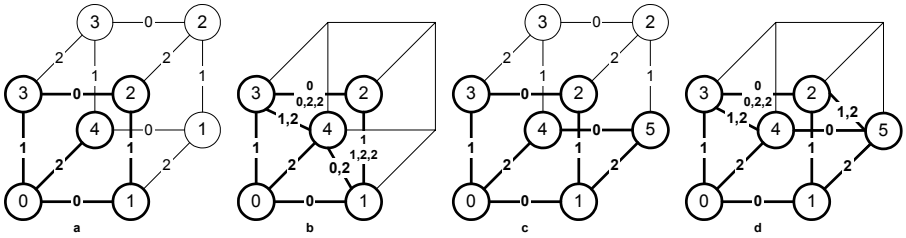

Figure 3. Network topology construction continued

Step d. Peer 4 arrives and contacts peer 0. Now, the network crosses a threshold – a hypercube with 2 dimensions cannot accommodate 5 peers, hence a third dimension is opened up. Peer 0 integrates peer 4 on its first vacant neighbor level as its new 2-neighbor. Peer 4 needs 3 neighbors, one on each neighbor level – but neither peer 0's 1-neighbor, peer 3, nor peer 0's 0-neighbor, peer 1, are linked to their own 2-neighbor which they could provide as a new neighbor to peer 4. Thus, peer 3 acts as temporary 1-neighbor for peer 4, whereas peer 1 acts as temporary 0-neighbor for peer 4, indicated once again by the link sets {0,2} and {1,2} among these peers (see Figure 3b). Figure 3a shows the existing peers in the network in bold style and the positions that are additionally covered by them in thin style. Once again, note that the positions that are additionally covered by peers determine the temporary connections the peers have to maintain, plus their edge labels. Figure 3a also demonstrates another basic rule: Peers that are 'closest' to a vacant position in the hypercube structure are always chosen to cover it. Here, 'closest' means that the peer on the highest neighbor level to a vacant position covers it. (In the more complicated case when a peer has to cover several positions, a peer covers the power set of its vacant neighbor levels – however, refer to [9] to find a detailed discussion.)

Among the other peers in the network, adding another dimension to the graph means the multiplication of existing links, too: For example, peers 1 and 2 could now both integrate 2-neighbors, which would then be linked on neighbor level 1. Thus, they tag their already existing {1} link additionally as {1,2,2} link. So do peers 2 and 3 with their already existing {0} link.

Step e. Peer 1 is contacted to integrate the newly arriving peer 5. Peer 1 is still lacking a 2-neighbor, thus peer 5 will be integrated on this position

(Figure 3d). Now, peer 1 can get rid of its {1,2,2} link to peer 2: It is moved to peer 5. However, since 2 is not peer 5's final 1-neighbor either, the link stays temporary: Peers 2 and 5 now maintain a {1,2} link among them. Peer 5 takes over peer 1's temporary {0,2} link to peer 4, which still lacks its final 0-neighbor. It has found one now, namely peer 5.

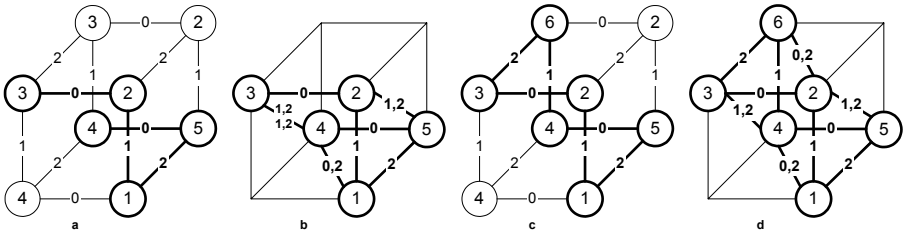

Figure 4. Network topology construction continued

Step f. Let us assume that peer 0 suddenly leaves the network. In the maintenance protocol, it is obliged to carry out a peer removal process: Basically, it decides which existing peers that it knows will be chosen to take over responsibility for the positions it gives up. In our example, peer 0 leaves only one position vacant, its original position in the graph – however, a node which covers multiple positions will have to find successors for each of its positions in the graph. Peer 4 takes over peer 0's position, establishing temporary links to the former neighbors of peer 0, peers 1 and 3. Figure 4a shows the new distribution of covering responsibilities, Figure 4b depicts the link structure that arises from this network state.

Step g. Peer 4 is contacted by peer 6 and decides to integrate it as its new 1-neighbor. This position is currently covered by peer 3, hence peer 4 forwards the integration control to peer 3, just as described in step c. In the example, all temporary links that are currently owned by peer 3, but originally belong to the new position of peer 6, are restored and passed on to peer 6. Additionally, peer 3 integrates peer 6 as its new 2-neighbor, arriving at Figure 4d. Note that both in step f and g a joining peer could have contacted any peer in the network without misbalancing the graph structure since all peers maintain temporary links.

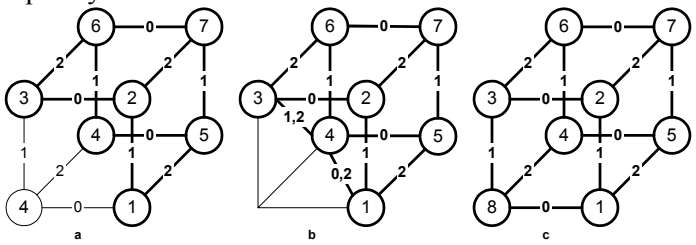

Figure 5. Network topology construction continued

Step h. Peer 6 is contacted by peer 7, leading to peer 7's integration as peer 6's new 0-neighbor. Figure 5a and Figure 5a b depict the state of the network: Almost all positions of a complete hypercube graph with 3 dimensions are held by active peers, only peer 4 still covers two positions in the hypercube.

What if several peers want to join the network simultaneously? We are currently working on turning our protocol into a real-time protocol, dealing with simultaneous node joins and departures. For example, parallel joins can be executed easily when join actions are time-stamped. We are studying the behavior of the protocol on a JXTA-based P2P infrastructure [6] [13].

Step i. On integrating peer 8, peer 4 pushes its links {1,2} and {0,2} to its new 2-neighbor, arriving at a complete hypercube topology again.

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 [9]): The node that is closest to the vanished node (in terms of a metric we call graph hop distance which uses the dimension order to compute a distance value between positions in the hypercube) 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 3.2.

3.2 Complexity

Assuming a relatively balanced graph structure, the algorithm as described above yields 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. To arrive at this complexity order, the algorithm uses optimizations not explained in detail in the walk-through in Section 3.1.

For example, the exact edge labels of temporary links do not have to be stored as a whole, they can be reconstructed from the missing neighbor levels of a peer when necessary (i.e. when this peer leaves the networks or integrates another peer). If a new hypercube dimension is opened up by integrating an additional peer (as has happened above in step d), this information is not broadcasted to all peers in the network – instead, it is propagated only when necessary, i.e. once again when nodes communicate on the issue of removing or integrating a peer. Networks that reach a large number of nodes can scale down again to a small number of nodes (as long as this takes place relatively balanced, see Section 3): Higher neighbor levels that are added during the

construction process are removed again if no peer in the network has any neighbor on a level any more. Once again, this information is not broadcasted in the network but locally inferred by every peer by observing its set of neighbor link sets it maintains.

4 Extending the Topology for Ontology-Based Routing

A HyperCuP empowered P2P network features good scalability and search times. However, additional knowledge might be available that can be used to further improve the P2P network performance: Often, information that peers are able to provide can be categorized as belonging to a general concept. Concepts can in turn be organized in a (global) ontology which defines the relationships between existing concepts. In such a case, we do not use only one hypercube for the whole network but construct one hypercube per concept which allows to reach all peers providing information or services for this concept very efficiently. Furthermore, the internal peer organization of a hypercube is structured such that the network can support logical combinations of ontology concepts in queries.

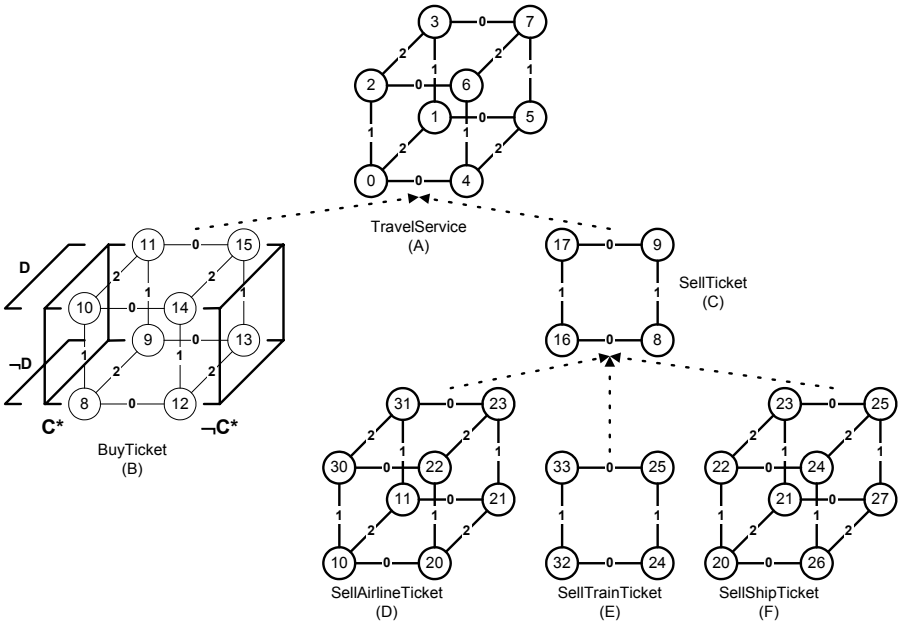

Figure 6. Ontology-structured network topology

4.1 Concept Hypercubes and Inter-Cluster Routing

For each concept in the ontology we construct exactly one hypercube which contains precisely those peers which directly support this concept (peers appear only in the most specific concept hypercubes).

A peer that supports several ontology concepts is a member of each hypercube that is associated with one of its supported concepts. In Figure 6, the network consists of six hypercubes, one per ontology concept. Hypercubes are linked with each other, the wiring among them mimics the is-a links in the global ontology. Between two connected hypercubes, the number of links equals the number of peers in the smaller hypercube, and the links are equally and deterministically distributed among the nodes in the larger hypercube.

To route concept-based queries, the peer picks one of the non-negated concepts in each single query and attempts to route the query to the hypercube associated with this concept. Routing to a destination hypercube associated with a concept is carried out via the inter-hypercube links. To prevent traffic hotspots in inter-cluster routing, concept hypercubes on one level in the global ontology (e.g. hypercubes B and C) are also linked with each other. This avoids the routing of all queries via the root concept(s) of the ontology.

4.2 Internal Hypercube Organization and Intra-Cluster Routing

A hypercube internally consists of groups of peers, i.e., peers that are on a common neighbor level. To answer queries consisting of logical combinations of ontology concepts, the internal structure of each hypercube reflects the concepts supported by its member peers. This is done by using partitions within the hypercube which group the peers supporting specific logical combinations of concepts. Queries are then forwarded to the partition which is ‘closest’ to the logical concept combination of the query. Each peer in a hypercube knows the internal organization of the hypercube, and it can reach every other group inside the hypercube within a logarithmic (to the size of the hypercube) amount of hops. As soon as the query has reached its destination group, it is broadcasted among all the peers in the partition. In Figure 6, only the internal concept assignment of hypercube B is depicted.

5 Related Work

Making P2P networks scalable has recently received much attention. In [12], 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 [10] such as CAN [7] and

Chord [11] 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 a topology to efficiently cluster peers in a P2P network which enables efficient broadcast and search algorithms without any message overhead at all during broadcast, logarithmic network diameter, and resiliency towards node failure. Super peers or central servers are not required. A global ontology is used to categorize peers as providers of particular information associated with ontology concepts to efficiently route queries. Organizing peers in this manner is especially useful in domains where data on a specific topic or query have to be collected from numerous peers in a P2P network, for example in the domain of semantic web services [4].

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] D. Krumme. Fast Gossiping for the Hypercube. *SIAM Journal on Computing 21: 2 365 – 380*, April 1992.

[3] Morpheus website. www.musiccity.com

[4] D. Martin et al. DAML-S: Semantic Markup for Web Services. White paper, 2001, www.daml.org/services/daml-s.

[5] 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.

[6] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. (2002). EDUTELLA: A P2P Networking Infrastructure based on RDF. In *Proceedings of the 11th International WWW Conference*, May 2002, Hawaii, USA.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, August 2001.

[8] McIlraith, S., Son, T.C. and Zeng, H. Semantic Web Services. In *IEEE Intelligent Systems. Special Issue on the Semantic Web. 16(2):46--53*, March/April, 2001.

[9] M. Schlosser, M. Sintek, S. Decker, W. Nejdl. Shaping up peer-to-peer networks. *Technical Report*, Stanford University, April 2002.

[10] 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.

[11] 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.

[12] 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.

[13] Project JXTA: An open, innovative collaboration. White paper, available at www.jxta.org.

[14] UDDI Technical White Paper. Available at www.uddi.org, September 2000.