

# HyperD: A Hypercube Topology for Dynamic Distributed Federated Databases

Andi Toce  
The Graduate Center (CUNY),  
365 Fifth Ave.,  
New York, NY 10016, USA

Abbe Mowshowitz  
The City College of  
New York (CUNY),  
Convent Ave. at 138th St.,  
New York, NY 10031, USA

Paul Stone  
Patrick Dantressangle  
Graham Bent  
IBM, Emerging Technology Services,  
Hursley Park, MP137,  
Winchester, Hants, SO21 2JN, UK

**Abstract**—In this paper we present HyperD, a dynamic distributed federated database where nodes are arranged to form a virtual hypercube topology. We describe a protocol designed to build and maintain the hypercube structure in a dynamic environment while preserving all desirable properties associated with hypercubes. We also show that HyperD is capable of implementing optimal hypercube routing and communication algorithms.

## I. INTRODUCTION

The popularity of dynamic fully distributed database systems has motivated many researchers to look for new ways to perform queries efficiently. Known topologies such as the hypercube have been found to be useful in both centralized and fully distributed systems. In a distributed environment however we are confronted with a series of new challenges. In particular, when deploying and maintaining a fixed structure in a dynamic environment we need to resolve issues such as scalability, churn, and fault tolerance while efficiently exploiting the benefits of the known topology.

Hypercubes are widely used in distributed systems due to a wealth of desirable properties. An  $n$ -dimensional hypercube is a  $n$ -regular graph of (relatively low) diameter  $n$ ; it is highly symmetric, having a transitive automorphism group, and thus allows for balancing the network load; it can be expanded easily to a higher dimension and is fault tolerant. In addition, a number of efficient communication algorithms exist for hypercubes.

This paper describes a protocol for building and maintaining HyperD, a Dynamic Distributed Federated Database (DDFD) [3] structured as a  $n$ -dimensional hypercube. We propose to organize all nodes belonging to a DDFD in a virtual complete hypercube. The new structure is scalable and fault tolerant. We deal with missing hypercube vertices by allowing neighboring nodes to gain temporary control of the vacant positions. In addition, we show that by doing so we are still able to make efficient use of known communication algorithms which take advantage of the hypercube structure.

## II. BACKGROUND AND RELATED WORK

A DDFD is a fully distributed relational database model for large networks of heterogeneous database systems. This architecture is well suited for a large range of database

applications where data is stored locally at each participating node while being accessible to all other nodes in the network. The choice of the hypercube as a model for our implementation is not coincidental. Hypercubes have been extensively used in parallel processing. An example can be found in [11]. A detailed survey of the use of hypercubes in parallel processing is found in [8]. Hypercubes have also found applications in sensor networks [6] and wireless communication networks [5], [7]. With the advent of peer to peer (P2P) networks, a variety of structured graphs including hypercubes have been used as a model for connecting peers. A hypercube P2P network is proposed in [15] where a distributed algorithm arranges the nodes in an implicit hypercube. A hypercube overlay network also affords the efficient use of a Distributed Hash Table (DHT) [2]. A DHT is a powerful abstract interface which can be used in publish and search algorithms. In [1] a hypercube-based P2P overlay for a DHT-based system is constructed allowing efficient lookup operations and providing a resilient structure against attacks and frequent network changes. A similar approach is also used in [12]. For a comprehensive look of the topological properties and formal definition of hypercubes we refer the reader to the following publications [10], [9], [14].

## III. PRELIMINARIES

The following notations and definitions will be used throughout this paper: Let  $D = (V_D, E_D)$  be a graph representing a HyperD instance with  $d = |V_D|$  nodes and  $e = |E_D|$  edges.  $D$  is taken to be a subgraph of the  $n$ -dimensional hypercube  $H = (V_H, E_H)$  with  $V_D \subseteq V_H$ ,  $E_D \subseteq E_H$ . Clearly,  $d \leq 2^n$  and  $e \leq 2^{n-1}n$ . Let  $D$  be a HyperD instance with  $d = |V_D|$  nodes and  $e = |E_D|$  edges.  $D$  is taken to be a subgraph of the  $n$ -dimensional hypercube  $H = (V_H, E_H)$  with  $V_D \subseteq V_H$ ,  $E_D \subseteq E_H$ . Clearly,  $d \leq 2^n$  and  $e \leq 2^{n-1}n$ .

### Definitions:

- 1)  $D$  is a *full hypercube* if  $V_D = V_H$  and  $E_D = E_H$ . Otherwise  $D$  is a *partial hypercube*.
- 2) The *Label Space*  $LS(H)$  is the set of binary labels of a hypercube  $H$ . If  $H$  is an  $n$ -dimensional hypercube,  $|LS(H)| = 2^n$ , the number of nodes in  $H$ .
- 3) The *Label Space*  $LS(v)$  of a node  $v \in D$  is the set of binary labels assigned to  $v$ . Note that a node in a

partial hypercube  $D$  may have to manage several labels to allow for simulating all the paths in a full hypercube.

- 4) Two distinct nodes in a full hypercube are adjacent if their labels differ in exactly one position. In this case we also speak of the labels being adjacent. Two distinct nodes  $u$  and  $v$  in a full hypercube are said to be  $i$ -neighbors,  $1 \leq i \leq n$ , if their binary labels differ only in the  $i$ th position.
- 5) Two nodes  $u \in V_D$  and  $v \in V_D$  are *neighbors* if there is at least one label in  $LS(u)$  that is adjacent to some label in  $LS(v)$ . Let  $i$  be the smallest value of the bit position for which a label of  $u$  is adjacent to a label of  $v$ . Then  $u$  and  $v$  are said to be  $i$ -neighbors.
- 6) Two labels are said to be joined by an *internal edge* if the two labels are adjacent and both belong to the same node. The *internal degree* of a label is the number of internal edges incident to the label.

To better understand all figures in this paper please refer to the appendix where symbolic representations are summarized and a detailed example is given.

#### IV. NETWORK PROPERTIES

HyperD has the following properties:

- HyperD is a set of interconnected database peers who form a single communication network without a centralized control in a pure ad-hoc fashion.
- Nodes in HyperD establish bidirectional connections which correspond to edges in the graph  $D$  modeling the network. Thus  $D$  is an undirected graph.
- Data is locally stored at each individual node but is accessible from any other node belonging to the network.
- HyperD is dynamic and nodes may enter or leave the network at any time. Connections in a HyperD are rearranged to deal with such changes in the network.
- HyperD approximates an  $n$ -dimensional hypercube if the number of nodes is between  $2^{n-1}$  and  $2^n$ .
- Each node  $v \in \text{HyperD}$  is assigned a subset of binary labels  $LS(v) \subseteq LS(H)$  where  $|LS(v)| \geq 1$ .
- Each label  $l \in LS(H)$  is assigned to a unique node in HyperD.
- Each node  $v \in \text{HyperD}$  knows the label set for each of its neighbors.
- Connections are periodically checked to detect possible connection or node failure. Other information can also be exchanged using these connection checks such as changes in the label set.
- If the number of nodes exceeds  $2^n$  HyperD expands to a virtual  $(n + 1)$ -dimensional hypercube.
- HyperD supports traditional hypercube routing and communication algorithms.

#### V. TOPOLOGY MAINTENANCE

Note that the initial deployment of HyperD can be seen in terms of node enters.

#### A. Node Enters

When a new node  $u$  intends to enter a HyperD network it first needs to contact any node  $n_0 \in \text{HyperD}$ . In turn  $n_0$  broadcasts the enter request to all other nodes in the network. The broadcast propagates along a hypercube minimum spanning tree with root at  $n_0$ . When a node  $n_i$  receives the enter request it responds to the broadcasting node if  $|LS(n_i)| > 1$  (it has an available label which can be assigned to the new node). Since the new node only needs to discover a single available label, if  $n_i$  responds to the broadcast it does not forward further the enter request to other nodes. This limits the broadcast range, thus saving valuable network resources. A node  $n_i$  will also respond if it is a leaf node in the hypercube spanning tree. If the leaf node has only a single label it sends a negative ("no label") response. A response from a leaf node indicates that no available labels were identified in that path connecting the root node to that particular leaf node. If all leaf nodes respond negatively, then HyperD is full (a complete hypercube) and no labels are available anywhere in the network. We summarize the enter steps below:

##### Enter Procedure

**Case 1:** HyperD is partial. (This is the most common scenario).

- 1) Node  $u$  broadcasts the enter request.
- 2) Nodes with more than one label respond to  $u$ .
- 3)  $u$  connects to the first responding node  $v$ .
- 4)  $u$  is assigned a label  $l \in LS(v)$  based on the following criteria: a)  $l$  is the label with the smallest internal degree. b) If more than one label have the same smallest internal degree choose the label with the largest binary value.
- 5) All connections from  $v$  relative to  $l$  are copied to  $u$ .
- 6) All connections from  $v$  which were exclusive to  $l$  are deleted from  $v$ .

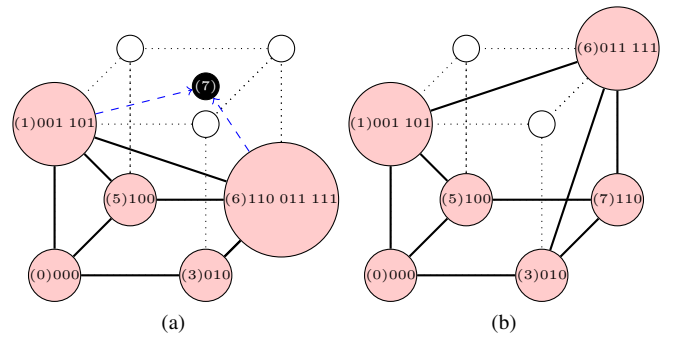


Fig. 1: Example of a node entering a partial 3-d HyperD

Figure 1 shows an example of a node entering a partial 3-dimensional HyperD. Node 7 broadcasts an enter request and receives a response from nodes 1 and 6 which have available labels in their label set (figure 1a). Assume node 7 accepts a label from node 6. Node 6 has three labels 110, 011 (with internal degree 1) and 111 (with internal degree 2). Node 6 chooses label 110 since it has the lowest internal degree and

its binary value is greater than 011. (7) connects to (3), (5) and (6). Node (6) disconnects from (5). The final result is shown in figure 1b.

**Case 2:** HyperD is full. (The probability of this scenario is small (approximately  $1/2^n$ ) and decreases as the network becomes larger).

- 1) Node  $u$  broadcasts the enter request.
- 2) All leaf nodes in the broadcast tree respond negatively since there are no available labels.
- 3) An "expand to next dimension" broadcast is sent. All nodes double their label set by appending a binary digit to the left of their current label  $l_i$ . Now each node has labels  $0l_i$  and  $1l_i$ . All previous connections remain unchanged.
- 4)  $u$  is assigned the largest of the two binary labels from the broadcast root node  $n_0$ .
- 5)  $u$  connects to  $n_0$  and all of its neighbors.

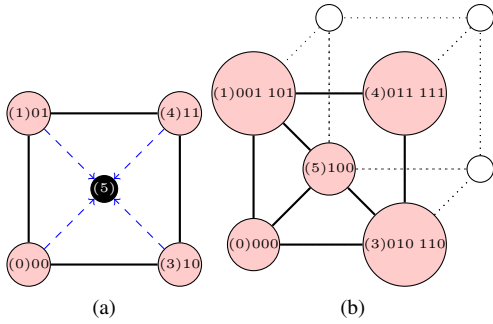


Fig. 2: Example of a node entering a full 2-d HyperD

Figure 2 shows an example of a node entering a full 2-dimensional HyperD. Node (5) broadcasts a enter request and receives a negative response from all nodes since there are no available labels to share (figure 2a). A message is therefore sent to all nodes to expand to a 3-dimensional hypercube by doubling their label set. Assume node (0) was the first node contacted by (5). Node (0) donates its largest binary label 100 to (5). Node (5) then connects to nodes (0), (1) and (3). The final result is shown in figure 2b.

### B. Node Departures

A node can leave the network at any time. Let  $v_1$  be a departing node.  $v_1$  may announce its departure and inform all neighboring nodes or  $v_1$  may fail or simply not announce its departure. In the latter case the departure is detected during any periodical connection check or during any network activity that involves the departed node. If a node detects the departure of its neighbor it notifies the node who will take control of the departing node labels based on the criteria described in the procedure below:

#### Departure Procedure

- 1) The departure of  $v_1$  is announced (or detected).

- 2) One of the neighboring nodes  $v_2$  assumes control of all labels from  $v_1$ .  $v_2$  is selected based on the following criteria:  $v_2$  has a label which is the lowest  $i$ -neighbor of any label associated with  $v_1$ . In the event two or more neighboring nodes satisfy the condition, the lowest  $i$ -neighbor with the largest binary value is selected.
- 3)  $v_2$  connects to all former  $v_1$  neighbors.

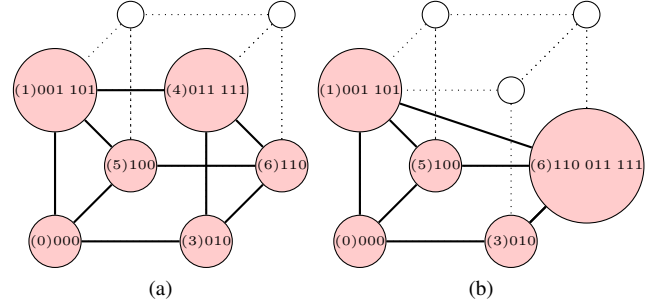


Fig. 3: Example of a node departing a HyperD

Figure 3 shows an example of a node departing HyperD. Node (4) leaves the HyperD network shown in figure 3a. Node (6) assumes control of the label set of (4) since (6) owns label 110 which is adjacent at position 0 to the largest label from (4) 111. Node (6) only needs to connect to (1) since it is already connected to the other former neighbors of (4). The final result is shown in figure 3b.

## VI. ROUTING AND BROADCASTING IN HYPERD

Broadcasting and pairwise communication are the most frequent network activities in a distributed system such as a DDFD. In this section we adopt known hypercube communication algorithms to work in a HyperD.

**Broadcast** (A single node needs to send a copy of the same message to all other nodes). Optimal algorithms exist to perform a broadcast in a  $n$ -dimensional hypercube  $H = (V_H, E_H)$  where exactly  $2^n - 1$  messages are needed. One approach is to use a minimum spanning tree with root at the broadcasting node. A number of different ways to construct a spanning tree exist for hypercubes [4]. A simple approach works as follows:

- 1) The broadcasting node sends the message to all its neighbors.
- 2) If a node  $v$  receives a message from its  $i$ -dimensional neighbor  $u$ , it forwards the message to all of its  $k$ -neighbors where  $i < k < n$ .

Figure 4 shows an example illustrating the broadcast procedure in a 4-dimensional hypercube with root at 1100. In a HyperD the broadcast algorithm is adopted with few modifications as follows:

- 1) The broadcasting node  $q$  selects a label  $l_q \in LS(q)$ .
- 2)  $q$  sends the message to all its neighbors relative to  $l_q$  (those neighbors having labels adjacent to  $l_q$ ). Each message includes the chosen label  $l_q$ .

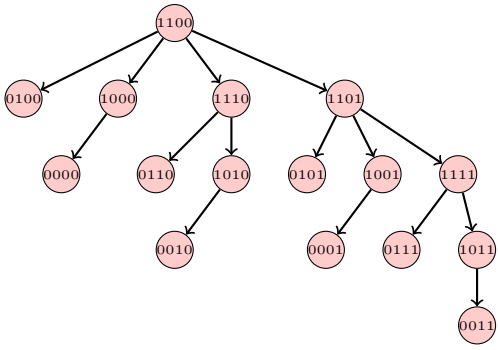


Fig. 4: Example of a broadcast in a 4-dimensional hypercube

- 3) For each other label in  $q$  adjacent to  $l_q$  repeat step 2.
- 4) If a node  $v$  receives a message from its neighbor  $u$  (including label  $l_u$ ), it forwards the message as follows for each of its labels  $l_v$  which are adjacent to  $l_u$ . Note that a node may have more than one label adjacent to  $l_u$ . ( $l_v$  is an  $i$ -neighbor of  $l_u$ ):
  - a)  $v$  forwards the message to all its  $k$ -neighbors of  $l_v$  where  $i < k < n$  including label  $l_v$ .
  - b) If  $v$  itself has a label  $l_w$  that is a  $j$ -neighbor of  $l_v$  where  $j > k$ , it sends a message to all its neighbors who have a label that is a  $g$ -neighbor of  $l_w$  where  $j < g < n$  including label  $l_w$ .

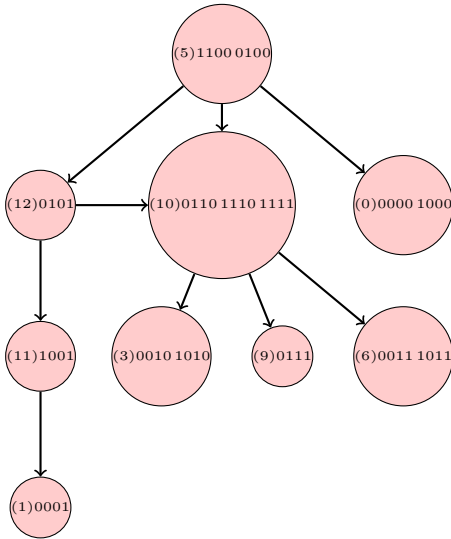


Fig. 5: Example of a broadcast in a 4-dimensional HyperD

Figure 5 shows an example of a broadcast in the partial 4-dimensional HyperD (figure 8d). The broadcast tree with root at node (5) is given. The broadcast in a partial HyperD is implicitly similar to a broadcast in a full hypercube. Since two or more labels may belong to a single node some of the spanning tree edges are internal to a node thus not producing messages. It may also occur that a node receives a message more than once. For example in figure 5 node (10) receives the message twice. In general, under normal circumstances

the number of redundant messages is expected to be a very small fraction of the overall number of broadcast messages.

**Unicast** (A node sends a message to a single node) The distance (shortest path) between two nodes in a hypercube is equivalent to the *Hamming Distance* defined as the bit difference between the two labels. If the distance between two nodes is  $k$  then there exist  $k$  shortest paths between the two nodes. This property is important in a dynamic network where nodes may fail during the process. If a path is temporarily unavailable an alternative path can easily be identified.

In a hypercube a unicast can be performed as follows: Let  $u$  be the sending node and  $v$  the receiving node. Let  $k$  be the distance between  $u$  and  $v$ . Let  $S$  be the set of bit positions where  $u$  and  $v$  labels differ. It follows that  $|S| = k$ . In  $k$  steps starting with  $u$  choose one of the bit positions  $i \in S$  and send the message to the  $i$ -neighbor  $w$ .  $w$  will be one hop closer to destination. Continue until  $v$  receives the message.

In HyperD the distance (shortest path) between two nodes  $u$  and  $v$  is no larger than the smallest bit difference between any pair of labels  $l_u \in u$  and  $l_v \in v$ . A unicast in a HyperD may be performed as follows: Let  $u$  be the sending node and  $v$  the receiving node. Let  $k$  be the smallest distance between  $l_u \in u$  and  $l_v \in v$ . Let  $S$  be the set of bit positions where  $l_u$  and  $l_v$  labels differ. It follows that  $|S| = k$ . In  $k$  steps starting with  $u$  choose one of the bit positions  $i \in S$  and send the message to the neighbor  $w$  having the label  $l_w$  which is adjacent to  $l_u$  at the bit position  $i$ .  $w$  will be one hop closer to the destination. Continue until  $v$  receives the message. Note that in some cases two labels belonging to the path traversed may belong to the same node thus no message is sent. In conclusion we can say that at most  $k$  steps are needed.

## VII. ANALYSIS

### A. Structure Properties

An  $n$ -dimensional hypercube is a well known regular graph with  $2^n$  vertices, a low diameter  $n$ , vertex degree  $n$  and average path length  $n/2$ . A  $n$ -dimensional HyperD is a subgraph of an  $n$ -dimensional hypercube  $H$ . The value of the diameter, vertex degree and average path length in a HyperD are not expected to significantly differ from those of  $H$ .

**Diameter** The maximum distance between a pair of nodes in HyperD does not exceed the number of digits of any label. The diameter of a  $n$ -dimensional HyperD with  $d = 2^n$  nodes is at most  $\log_2 d + 1$  if  $2^{n-1} \leq d \leq 2^n$ . The diameter is  $\log_2 d + 1$  when the structure has just expanded from a  $(n-1)$ -dimensional to a  $n$ -dimensional hypercube. The diameter is  $\log_2 d$  if HyperD is full.

**Vertex Degree** The maximum node degree for a node  $v \in \text{HyperD}$  is  $|LS(v)|\log_2 d$ . The average node degree is expected to be  $\log_2 d + 1$ .

**Path Length** The average path length for  $D$  is  $\log_2 d/2 + 1$ . In general, the diameter, vertex degree and average path length in a HyperD are all in the order  $\log d$ .

## B. Maintenance Cost

**Node Enters** As described in section V-A when a node enters a HyperD it follows the following steps: (1) broadcast the enter request; (2) receive responses; (3) expand to the next dimension (if no available labels); (4) connect to a subset of nodes in HyperD. We look at the cost of performing each step. In step (1) the broadcast in the worst case may need to reach all nodes in the network if HyperD is full and there are no labels available to share. The best case scenario occurs when the first contacted node has already an available label, thus there is no need to forward the message further. In most cases, the broadcast will reach only a subset of nodes.

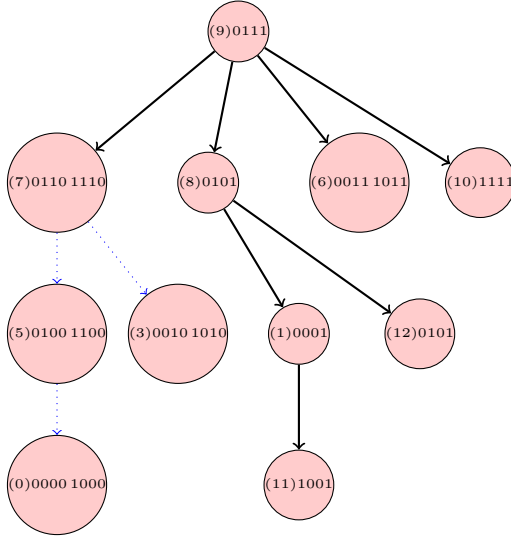


Fig. 6: Example of a node enter broadcast in a partial 4-dimensional HyperD

Figure 6 shows the complete broadcast tree with root at node (9) in a partial 4-dimensional HyperD. The solid directed lines (  $\longrightarrow$  ) represent edges utilized during a enter broadcast with root at node (9). The dotted directed lines (  $-\ - - \longrightarrow$  ) show edges from the complete broadcast tree not utilized during the enter broadcast.

In step (2) the number of nodes that respond to an enter broadcast is at most the number of leaf nodes in the complete broadcast tree. This occurs when HyperD is full. In most cases a smaller subset will respond. In figure 6, five nodes ( (6), (7), (10), (11) and (12) ) respond to the enter request.

Step (3) only takes place when the HyperD is full. All nodes need to be notified that the network will expand to a higher dimension. The number of messages is equivalent to a complete broadcast and is approximately  $d$ . Since the expansion occurs only when HyperD becomes full (the number of nodes doubles) the cost is a very small fraction in the overall maintenance cost.

Step (4) Each new node has to establish at most  $n = \log_2 d$  connections. Note that a neighboring node may own multiple labels which are adjacent to the new node. In such a case the number of connections is fewer than  $n$ .

**Node Departures** Once the departure of a node  $v$  is announced or detected one of its neighboring nodes takes control of all labels in  $LS(v)$  as described in section V-B. The new "owner" connects to all previous neighbors of  $v$ . At most  $|LS(v)|(\log_2 d - 1)$  are needed if all neighbors are distinct. Note that a neighbor may own multiple adjacent labels. In such cases the number of connections will be smaller.

## C. Query Efficiency

A hypercube-based network such as HyperD offers a number of savings and advantages in distributed query processing and optimization. Broadcasting and routing in a hypercube structure are cheaper in terms of the number of messages sent. The network load is shared among nodes minimizing bottlenecks and failures. A hypercube allows for parallel processing of data and offers multiple disjoint routing paths between any pair of nodes in the network. In addition, pairwise distances can easily be estimated in a hypercube allowing a query optimizer to select a better query execution plan in those cases where the use of joins and semijoins is prevalent [13]. Overall, the additional cost of maintaining a fixed structure is likely outweighed by the savings produced.

## VIII. CONCLUSIONS AND FUTURE WORK

In this paper we presented HyperD, a hypercube DDFD. A virtual complete hypercube is implicitly maintained by allowing nodes in the network to manage more than one position in the hypercube. We also show that by maintaining a virtual complete hypercube we are able to take advantage of known communication algorithms. The HyperD structure produces savings especially in those networks where churn rate is small relative to the overall network activity. In addition, the network is more resilient against failure and bottlenecks.

Possible directions for future research may include dealing with high churn levels by allowing nodes to operate at different dimensions while preserving efficient communication capabilities. Moreover, a DHT-based overlay can be constructed using the HyperD model.

## IX. ACKNOWLEDGMENTS

Research was sponsored by the U.S. Army Research Laboratory and the U.K Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defense or the U.K Government. The U.S. and U.K. Governments are authorized to reproduce and distribute for Government purposes notwithstanding any copyright notation hereon.

## REFERENCES

- [1] E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro. Peercube: A hypercube-based p2p overlay robust against collusion and churn. In *SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 15–24, Washington, DC, USA, 2008. IEEE Computer Society.









- [2] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.
- [3] G. Bent, D. Dantressangle, A. Mowshowitz, and V. Mitsou. A dynamic distributed federated database. In *Proceedings of the Second Annual Conference of ITA*, 2008.
- [4] Dimitri P. Bertsekas, C. Özveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis. Optimal communication algorithms for hypercubes. *J. Parallel Distrib. Comput.*, 11(4):263–275, 1991.
- [5] Chao-Tsun Chang, Chih-Yung Chang, and Jang-Ping Sheu. Bluecube: constructing a hypercube parallel computing and communication environment over bluetooth radio systems. *J. Parallel Distrib. Comput.*, 66(10):1243–1258, 2006.
- [6] Po-Jen Chuang, Bo-Yi Li, and Tun-Hao Chao. Hypercube-based data gathering in wireless sensor networks. *J. Inf. Sci. Eng.*, 23(4):1155–1170, 2007.
- [7] Hongwei Huo, Wei Shen, and Youzhi Xu. Virtual hypercube routing in wireless sensor networks for health care systems. *Chinese journal of electronics*, 19(1):138–144, 2010.
- [8] S. Lakshmivarahan and Sudarshan K. Dhall. Ring, torus and hypercube architectures/algorithms for parallel computing. *Parallel Comput.*, 25(13-14):1877–1906, 1999.
- [9] Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., California, USA, 1992.
- [10] Ted G. Lewis. *Network Science, Theory and Application*. A John Wiley and Sons, Inc., New Jersey, USA, 2009.
- [11] Youyao Liu, Jungang Han, and Huimin Du. A hypercube-based scalable interconnection network for massively parallel computing. *JCP*, 3(10):58–65, 2008.
- [12] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 3–11, Washington, DC, USA, 2006. IEEE Computer Society.
- [13] A. Mowshowitz, A. Kawaguchi, A. Toce, A. Nagel, G. Bent, P. Stone, and P. Dantressangle. Query optimization in a distributed hypercube database. In *Proceedings of the Fourth Annual Conference of ITA*, 2010.
- [14] Youcef Saad and Martin H. Schultz. Topological properties of hypercubes. *IEEE Trans. Comput.*, 37(7):867–872, 1988.
- [15] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. Hypercup - hypercubes, ontologies and efficient search on p2p networks. In *LNCS*, pages 112–124. Springer, 2002.





## APPENDIX
















































































Figures 7 and 8 in this appendix demonstrate the creation and growth of a HyperD network. 13 nodes enter to form a 4-dimensional HyperD. In the process 3 of those nodes depart the network.

### Legend:

-  is node  $u$  in HyperD with label set  $LS(u)$ .
-  is a new node  $u$  entering HyperD.
-  is a vacant position in the underlying hypercube structure.
-  is an edge connecting two nodes in HyperD.
-  is an unused edge in the underlying hypercube.
-  represents a response message to a enter broadcast.

### Example Description:

- Part 7a: HyperD consists of a single node .
- Part 7b: Node  wants to enter. HyperD expands to the 1st dimension.
- Part 7c:  connects to  to form a full 1-dimensional hypercube.

- Part 7d: Node  wants to enter. HyperD expands to the 2nd dimension since there no available labels.
- Part 7e: Node  Get label 11 from node  which is the first node to respond to the enter request. Node  connects to  and .
- Part 7f: Node  wants to enter. Node  is the only node to respond.
- Part 7g: Node  gets label 10 and fills the last position in the hypercube.
- Part 7h: Node  departs. Node  takes over the label from  since it is the lowest dimension neighbor.  connects to .
- Part 7i: Node  wants to enter. Node  is the only node to respond.
- Part 7j: Node  gets label 11 and fills the last position in the hypercube.
- Part 7k: Node  wants to enter. HyperD expands to the 3rd dimension since there no available labels.
- Part 7l: Node  gets the largest label 100 from the first responding node .  connects to ,  and .
- Part 7m: Node  wants to enter. Node  is the first to respond.
- Part 7n: Node  gets the largest label 110 from  and connects to ,  and .
- Part 7o: Node  departs. Node  takes over the label set since it is the lowest dimension neighbor relative to the largest label 111 from .  connects to .
- Part 7p: Node  wants to enter. Node  responds first.
- Part 7q: Node  donates label 110 which is the largest label with internal degree 1.  connects to ,  and . Node  disconnects from .
- Part 7r: Node  wants to enter. Node  responds first.
- Part 7s: Node  gets the largest label 101 from  and connects to ,  and .
- Part 7t: Node  wants to enter. Node  is the only one to respond.
- Part 7u: Node  gets the largest label 111 and fills the last vacant position in the 3-dimensional hypercube.
- Part 8a: Node  enters HyperD. Since the network was a full 3-dimensional hypercube it now expands to a 4-dimensional hypercube. The first to respond is node  which donates its largest label 1111. Node  then connects to nodes , ,  and .
- Part 8b: The new node  gets label 1001 from node  and connects to nodes , ,  and .
- Part 8c: The new node  gets label 1101 from node  and connects to nodes , ,  and .
- Part 8d: Node  departs. Node  takes control of all labels since its the smallest dimension neighbor relative to 's largest label 1110. Node  connects to .

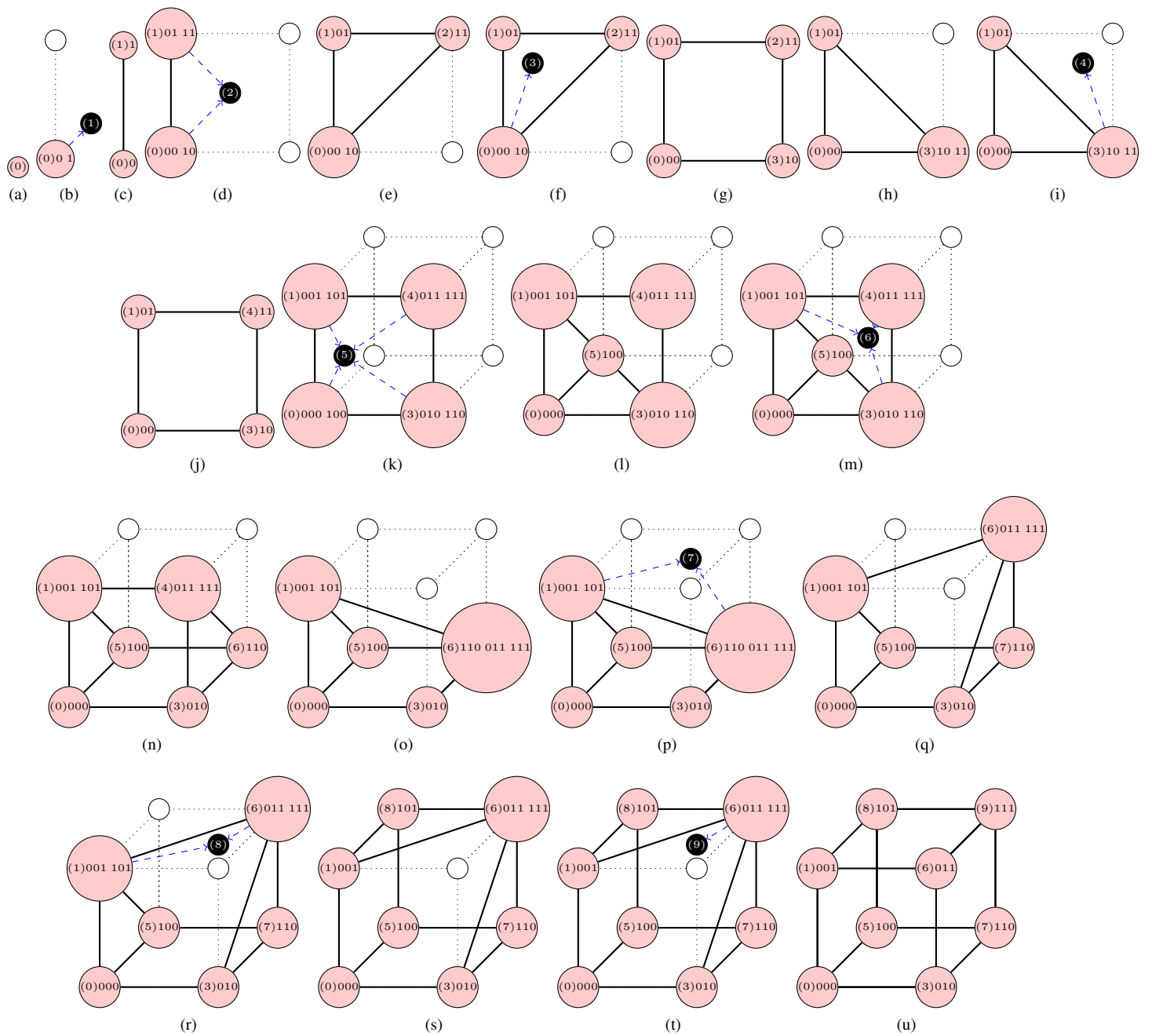
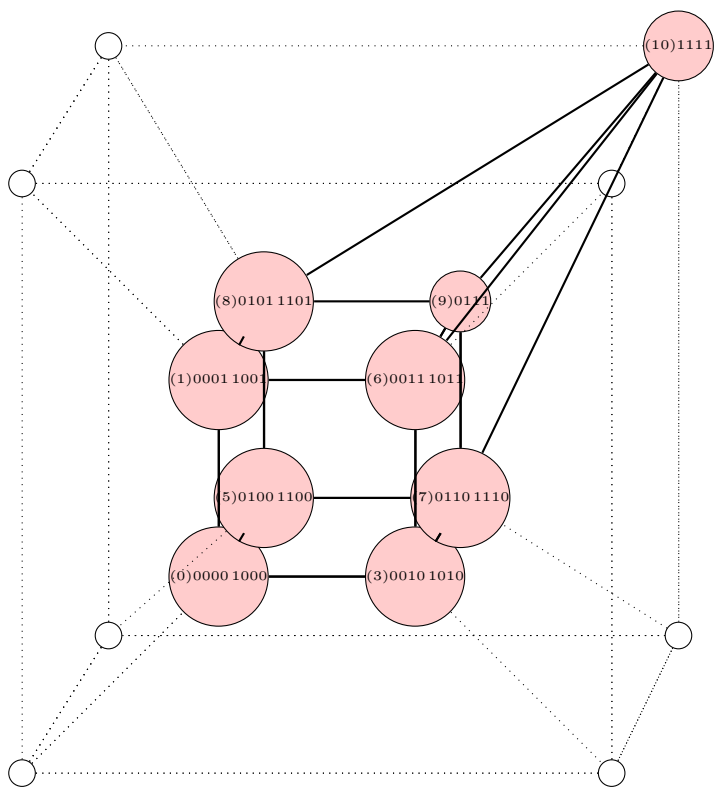
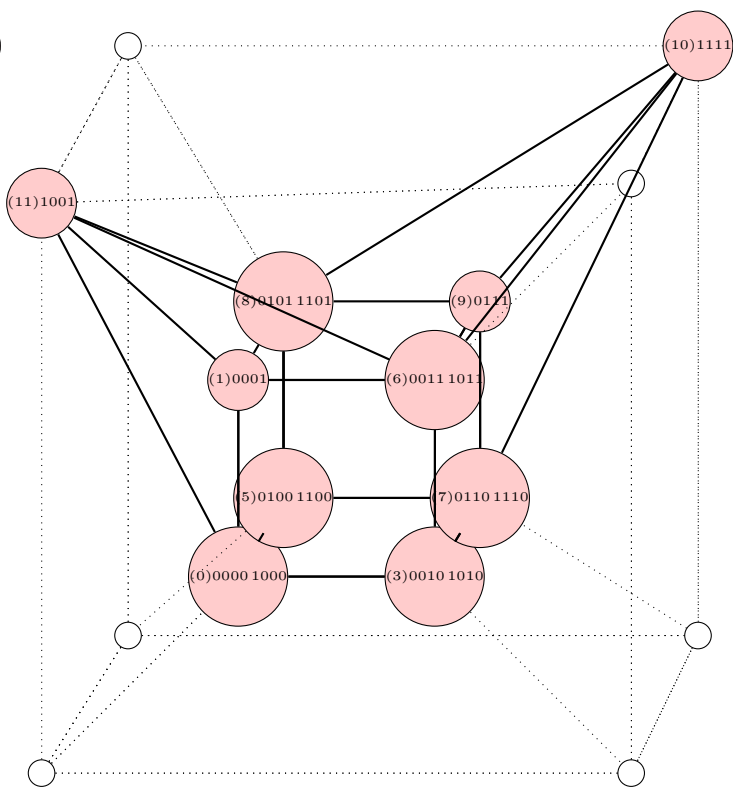


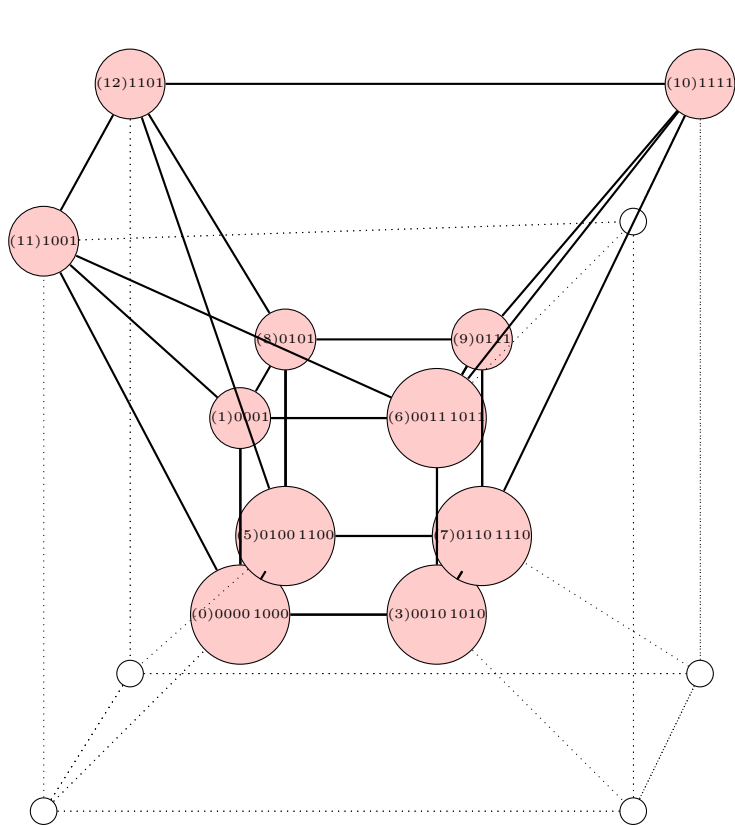
Fig. 7: HyperD Growth Example Part 1



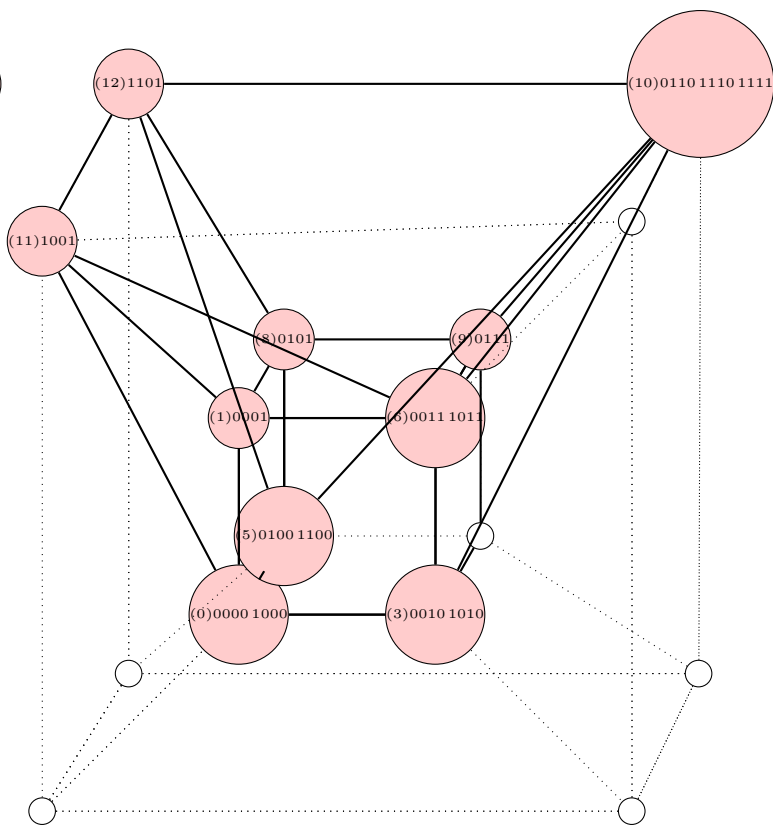
(a)



(b)



(c)



(d)

Fig. 8: HyperD Growth Example Part 2