EFFICIENT COMMUNICATION THROUGH STRUCTURED NODE LABELING IN PEER-TO-PEER NETWORKS

by

ANDI TOCE

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2013

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Abbe Mowshowitz

Date | Chair of Examining Committee

Theodore Brown

Date | Executive Officer

Akira Kawaguchi

Theodore Brown

Graham Bent

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

EFFICIENT COMMUNICATION THROUGH STRUCTURED NODE LABELING IN PEER-TO-PEER NETWORKS

by

ANDI TOCE

Adviser: Professor Abbe Mowshowitz

Peer to Peer (P2P) networks have become increasingly popular in recent years as administrators are choosing to move away from monolithic centralized networks. These networks, while offering significant advantages in a variety of applications, are presented with a new set of challenges, from communication efficiency to network vulnerability and security. Numerous researchers have proposed a variety of solutions to improve the quality of service in P2P networks. A critical factor in maintaining an acceptable level of service quality is the efficiency and reliability of communication among nodes. One way of improving communication is to assign identifiers (labels) to each node and then use these labels to facilitate and improve message routing. This proposed work introduces two labeling schemata for P2P networks. Each participating node is assigned a label set. Labels are then used to determine node positions within an engineered logical overlay and identify routing paths during communication. We prove that the assignment of labels reduces the overall cost of communication thus saving valuable network resources. These theoretical findings are confirmed by experimental results with randomly selected P2P networks of various sizes. Detailed statistics on the performance of each protocol are provided which show clearly the practical utility of the labeling approach.

Acknowledgements

This work has come to fruition due to the help and support of a number of collegues and friends, who in different ways have inspired me and made this work possible.

First of all, I would like to thank and express my gratitude to my advisor and coauthor, professor Abbe Mowshowitz. His care, help, encouragement, experience and expertize have played a crucial role in the complition of this dissertation. I also want to thank all members of the advising committee, professor Akira Kawaguchi and professor Ted Brown for their continuous support. Professor Brown and professor Amotz Bar-Noy were the first to include me in a research project at the Graduate Center as part of the International Technology Alliance (ITA). This work led to a long and fruitful cooperation with my subsequent ITA project on distributed database systems. I have had the privilegde to work on this project with professor Mowshowitz, professor Kawaguchi, as well as Andrew Nagel from the Graduate Center (CUNY), Paul Stone, Patrick Dantressangle and Graham Bent from IBM-UK. This experience, more that anything else, has been fundamental for my professional growth and has provided me with the necessary background needed for this work. Special thanks go to Graham Bent who has graciously agreed to serve as the outside member of the dissertation committee.

I also want to thank my colleagues and friends at LaGuardia Community College, in particular Dr. Joyce Zaritsky, who have given me enough support and flexibility to continue my graduate studies while providing for my family.

This research was sponsored in part by the U.S. Army Research Laboratory and the U.K Ministry of Defense under Agreement Number W911NF-06-3-0001.

Most importantly, I want to thank my family, my parents for their unconditional love, for believing in me and always pushing me to aim higher, and of course my wonderful wife

Sandra and our two beautiful boys, Arjan and Alek, who are the greatest inspiration for everything that I do.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Peer-to-Peer Networks

A Peer-to-Peer (P2P) network is a network of computer devices (nodes) who connect to each other without any centralized control in a purely ad-hoc fashion. During the existence of a P2P network new nodes may enter the network, existing peers may leave the network at any time or may change their position (location) within the network. Each node has equal rights within the network and is expected to not only benefit from the services offered but to also offer its full cooperation.

P2P networks have seen an increasing number of applications in recent years. P2P networks are increasingly popular because of a number of advantages that they offer. Connected peers share any available resources from content files to computer storage and processing power. A P2P network is more flexible, allowing for frequent network changes, it is more robust given the absence of a central authority and a single point of failure. On the other hand, P2P networks are faced with a number of problems such as network scalability, content search, content delivery, network security and reliability. New approaches are needed to deal with these issues.

1.2 Motivation

P2P networks despite their dynamic nature are expected to offer an acceptable degree of service quality. Any implemented P2P network protocol should be able to: (1) maintain connectivity; (2) self-organize without the need of any central control or authority; (3) deal efficiently with any network changes such as new node arrivals, node failures, node deletions and node mobility; (4) guarantee an accurate and reliable exchange of information between nodes; (5) be able to support the exchange of large amounts of information between nodes in a reasonable amount of time; (6) be scalable adjusting to the growth of the network; (7) and be resilient against attacks or failure.

However, it is quite challenging to achieve all of the above goals, particularly in the absence of a central authority to coordinate these efforts. A large body of work is dedicated to improving performance of P2P networks. Extensive research is conducted on various aspects of such networks in a quest to improve the overall quality. A crucial aspect of P2P networks that directly impacts the overall performance is the way nodes communicate, search for content and find and maintain routes. Reducing the communication load for performing typical network activities benefits the network in a variety of ways. Finding shorter routes and reducing the number of messages exchanged during network operations increases reliability, decreases the time required for completion, reduces resource consumption such as processing power, battery life, bandwidth utilization and reduces the likelihood of network fragmentation and node failure.

1.3 Contribution

This proposed work introduces two labeling schemata for P2P networks. The first labeling schema, HyperD, assigns to each participating node a label set. These labels are then

used to determine which nodes will form a virtual connection. A virtual overlay is formed using these virtual connections. Each label represents a position in a virtual hypercube, therefore nodes in the virtual network connect to form a hypercube topology. In order to maintain a complete hypercube structure, nodes may temporarily assume control of more than one hypercube position (i.e., they are assigned multiple labels). Finally, labels are used to facilitate communication between nodes adopting known optimal hypercube communication algorithms.

Hypercubes are well known structures which have been extensively studied and used in distributed and parallel systems. Assigning labels to nodes when building a structured overlay is not new. Typically these labels are assigned using a random process, therefore the position of each node in the network is also determined randomly. Quite often this produces an overlay that differs substantially from its underlying physical network, therefore theoretical results could be very different from the actual real cost of communication. In other cases, labels are assigned using the recursive property of hypercubes or following a hierarchical structure. Our work differs from these previous approaches in that we preserve the integrity of the hypercube by allowing nodes to take control of multiple labels. This requires additional maintenance overhead but permits us to optimally adopt hypercube communication algorithms. Furthermore, we employ a process of preferential attachment to assign labels to new arriving nodes. Each assignment is done using a fitness function which increases the probability that two virtual nodes are also neighbors (or at least within a small number of hops) in the underlying physical network. In addition, we explore a label assignment variant which limits the label search to only a small constant number of hops. By doing so we significantly improve the virtual-physical alignment while preserving the hypercube structure and the efficiency of communication.

The second schema, Binary Labeled Wireless Network (BLWNet), also uses binary labels

as node identifiers. Unlike with HyperD, labels are only used to determine paths during routing and broadcasting. No virtual connections are formed. Nodes communicate directly with their physical neighbors (nodes within transmission range). This choice is made in order to avoid any potential misalignment between the physical network and the virtual overlay. Network changes such as nodes entering and leaving the network and the corresponding label assignments are handled using only local information. During network activities such as broadcasting and routing, labels are used to determine next hop forwarding only based on the shared knowledge among neighboring nodes. Using only local information is crucial to minimize the impact of each activity and reduce the cost of communication. The choice of applying this schema to wireless ad-hoc networks is motivated by the need for a specific physical network which helps with protocol implementation and experimental setup. Results however can easily extend to more general P2P network models. In wireless ad-hoc networks nodes typically follow either a proactive or reactive approach to identifying routing paths. BLWNet features characteristics from both these routing methods. A node using the BLWNet protocol can proactively store information about other nodes in the network. However, only node labels are needed to be stored instead of the entire routing path information. A routing path is determined reactively when needed by using the available label information.

We show that both labeling schemata reduce the overall cost of communication thus saving valuable network resources. These theoretical findings are confirmed by experimental results with randomly selected P2P networks of various sizes. Detailed statistics on the performance of each protocol are provided which show clearly the practical utility of the labeling approach.

1.4 Organization

In chapter 2 we provide an overview of P2P networks and summarize the various approaches used in the literature to design overlay networks. We focus on hypercube networks which serve as models for our proposed labeling schemata. We provide some necessary background on hypercube properties and algorithms. We also provide a brief survey of the different approaches used to build and maintain hypercube topologies. Chapter 3 introduces HyperD. We outline HyperD properties, provide details on how a HyperD network is initially deployed and then subsequently maintained. We also summarize results of our experiments. Chapter 4 introduces BLWNet and all related implementation details. In addition, background information is added about Wireless ad-Hoc networks, the most common challenges they face and the various approaches that are proposed to improve communication in these networks. We also give detailed experimental results validating some of the properties of the network. A summary of findings and conclusions is given in chapter 5. Finally we add four appendices (A-D) which include larger and more detailed examples and some of the data sets from these experiments.

Chapter 2

Background

2.1 Peer-to-Peer Overlays

In a typical P2P network nodes form an overlay to facilitate data discovery and improve communication. An overlay network is an abstract architecture usually implemented at the application layer which is build on top of the existing physical layer. Nodes belonging to this overlay form virtual connections. Two nodes are virtual neighbors in the overlay if they know each other and directly communicate using the underlying physical network. In practice two nodes which are virtually connected may not necessarily be adjacent in the actual physical layer. Figure 2.1 shows an example of a small P2P network and the corresponding overlay. The lower layer shows physical connections among nodes and the upper layer shows all virtual connections these nodes have established to form the overlay. Note that in some cases two nodes have a virtual connection even though they are two hops apart in the actual physical layer. In practice, two distant nodes may connect virtually. When virtual connections do not correspond to physical connections we speak of the two layers being misaligned. Depending on the overlay protocol we can have various

degrees of misalignment. As a consequence, a good solution over the virtual connections may not necessarily translate into a good solution in the actual physical network.

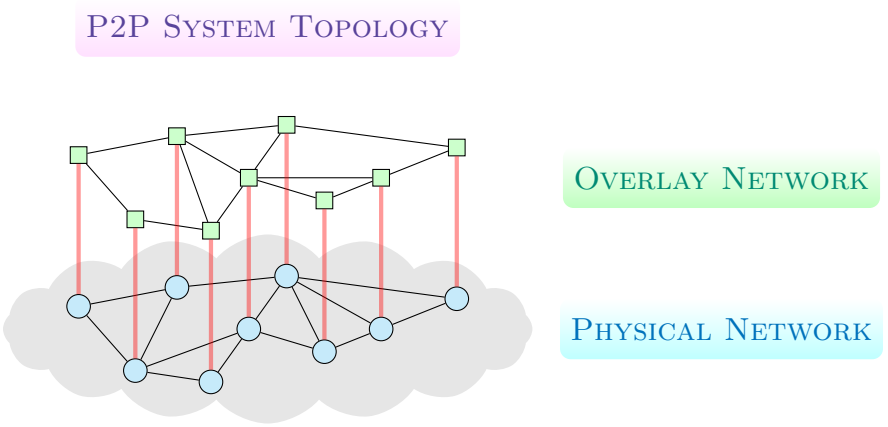

Figure 2.1: Example of an Overlay Network

A large number of papers have been written on P2P networks and many types of overlay networks have been proposed. It is beyond the scope of this thesis to review the extensive literature in detail. Several surveys are available including [1, 2]. Here we provide a brief summary of recent results relevant to the problem at hand. P2P overlays are commonly categorized as structured and unstructured. In unstructured P2P networks nodes connect using a random process, usually using only adjacent members. In some cases nodes may connect based on their content or common features and interests. The resulting overlay graph does not have a fixed structure and quite often resembles a random graph or a scale-free graph. These types of networks normally operate using a very small set of rules and are easier to maintain but lack in efficiency and scalability. The absence of a known structure forces nodes to use expensive search methods such as flooding and random walks. In structured P2P networks, the resulting overlay graph approximates a known graph structure or geometry. Labels (keys, identifiers, addresses) are used to determine connections. Maintaining the chosen structure requires some additional overhead but communication and content discovery is usually much more efficient. Hybrid

approaches also exist. Figure 2.2 adapted from [1] provides a very good classification of such approaches.

2.2 Hypercube Networks

Hypercubes have been used in a large variety of applications due to a wealth of desirable properties. Hypercubes are conceptually simple, regular, symmetric graph structures with relatively low diameter and moderate node degree. A number of efficient optimal communication algorithms exist for hypercubes, from identifying paths between participating nodes to broadcasting messages to all nodes in a network. These properties make hypercubes an excellent choice for modeling distributed networks and minimizing the cost of network operations.

This chapter gives the necessary background information. A number of parallel and distributed applications are summarized justifying the choice of hypercubes as a model for the logical overlay in a P2P network. The binary hypercube or simply the hypercube is defined and some notations used in this document are introduced. Most important, we list relevant hypercube properties and emphasize their application to P2P networks. We also analyze and compare different approaches in recent literature for deploying and maintaining hypercube structures. Finally, we summarize a number of optimal communication algorithms specific to hypercube networks and show their importance in creating a more efficient, robust and resilient P2P network.

2.2.1 Hypercube Definitions and Properties

A k-dimensional hypercube Q_k also referred to as k-cube, binary k-cube or boolean k-cube is an abstract structure containing $n = 2^k$ nodes. $Q_k(V, E)$ is a special type of

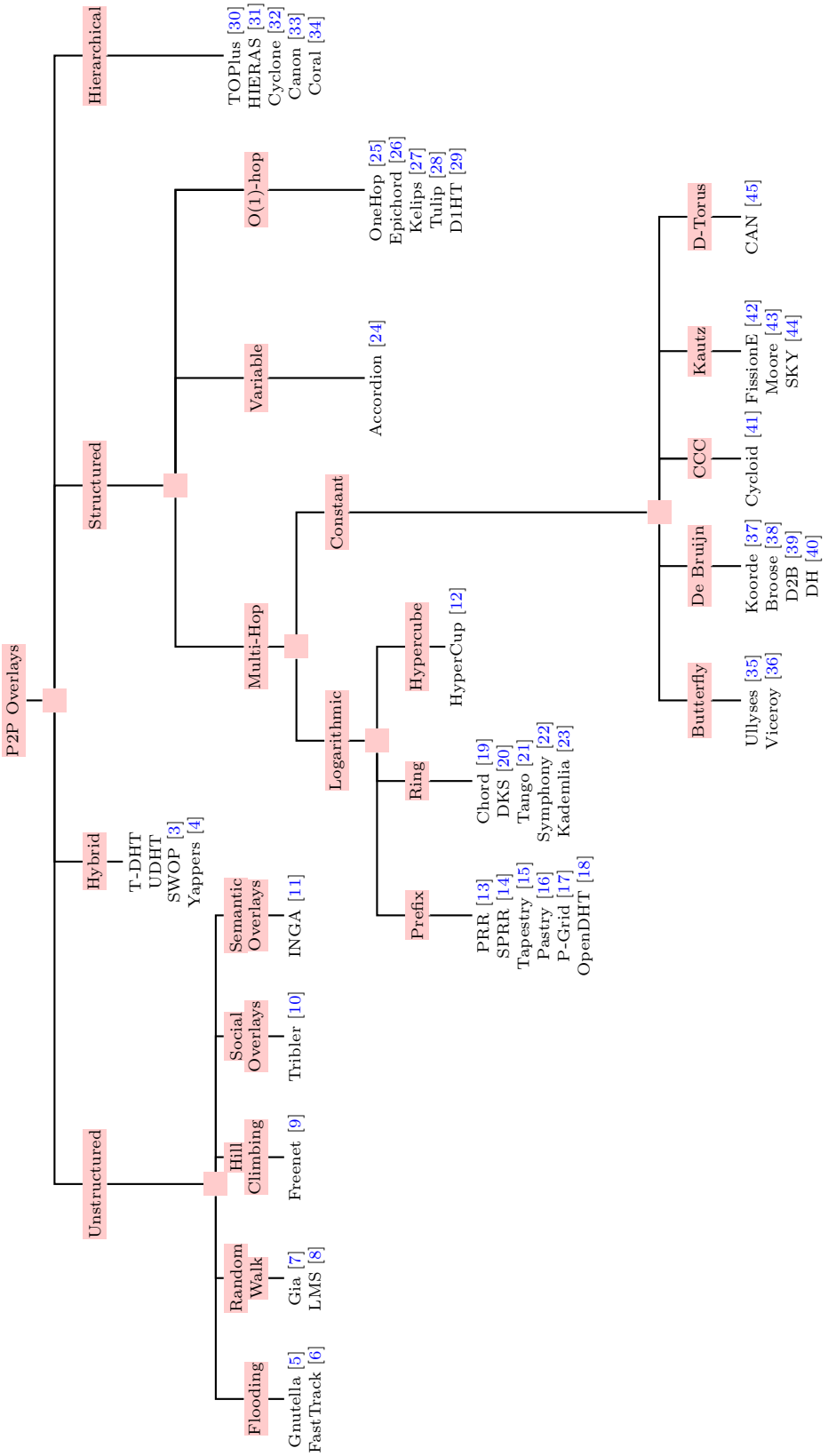

FIGURE 2.2: P2P Classification (adapted from [1])

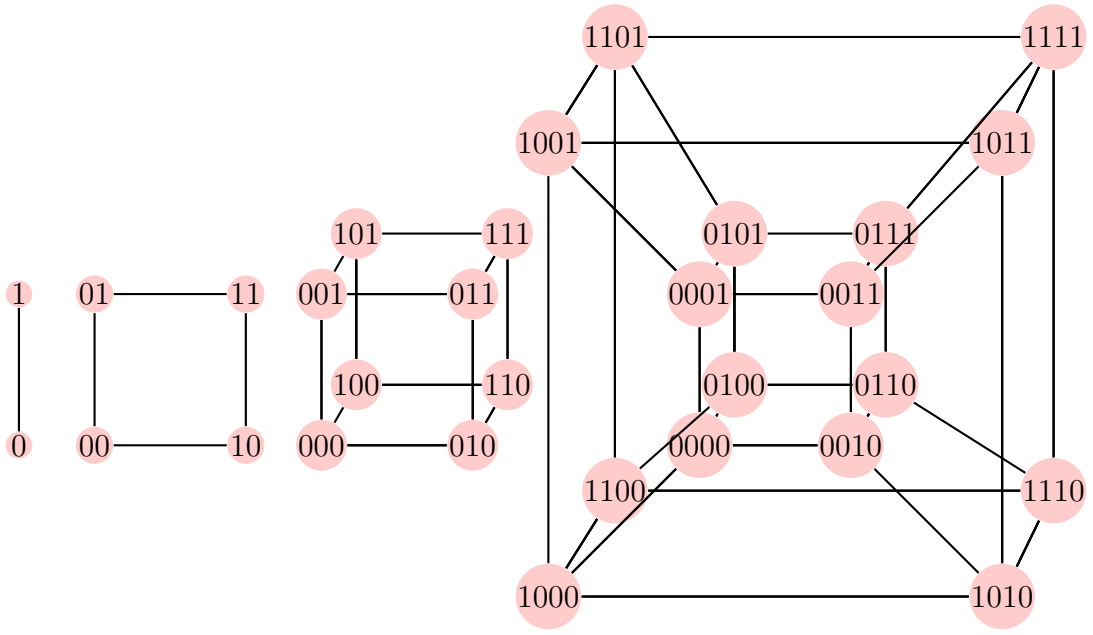

FIGURE 2.3: Examples of Binary Hypercubes for Dimensions 1 through 4

graph where V is the set of nodes (vertices) and E is the set of edges. Each node $v_i \in V$ is assigned a unique k-binary digit label (identifier). An edge $e_i \in E$ is then used to connect any two nodes whose binary labels differ at exactly one digit. A total of $k2^{k-1}$ edges are used. It follows that $|V| = n = 2^k$ and $|E| = k2^{k-1}$. Figure 2.3 shows examples of hypercubes from $k = 1$ to $k = 4$.

Recursive Hypercube Definition:

Let Q_k be the k-th dimensional hypercube and K_n be the complete graph of n nodes. Then a hypercube can recursively be defined as follows:

$Q_1 = K_2$

$Q_{k+1} = Q_k \times K_2$

where the operator $\times$ denotes the graph cartesian product.

Figure 2.4 shows the recursive transition from Q_2 to Q_3 based on the above definition.

Notation:

Below is the list of some notations used in this chapter.

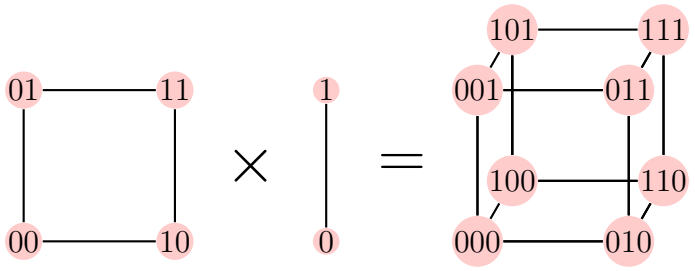

FIGURE 2.4: Recursive Definition Example: $Q_2 \times K_2 = Q_3$

$Q_k \rightarrow k$-dimensional hypercube.

$k \rightarrow$ the dimension of the hypercube / the hypercube diameter / node degree.

$n \rightarrow$ the number of nodes in a hypercube P2P network. It follows that $n = 2^k$.

$p \rightarrow$ the minimum path size between two nodes in Q_k.

Below we list those properties that make hypercube structures a desirable choice for modeling a P2P network.

Property 2.1. *Q_k is a regular graph with transitive automorphism group.*

Property 2.2. *All nodes in Q_k have the same eccentricity (the maximum distance between the node and any other node in Q_k).*

Remark 2.3. Properties 2.1 and 2.2 imply that all nodes and edges in a hypercube are equally important within the structure. In general, no node or edge in a hypercube can be distinguished from the other nodes or edges based on their properties. Given such a symmetry and regularity we expect that the load of the network is fully balanced under normal conditions, meaning all nodes and edges share the network load equally. This makes the network more resilient against failure by avoiding bottlenecks. These properties are also useful in designing optimal communication algorithms.

Property 2.4. *The diameter (the largest distance between any two nodes) of Q_k is k.*

Remark 2.5. The diameter of a network plays an important role in the cost of communication. We think of the cost in terms of the number of messages needed to perform a

network activity. In turn, fewer forwarded messages will also result in a faster communication time and decrease the probability of node overload and failure. A logarithmic order diameter such as in the case of the hypercube guaranties that a message exchanged between any two nodes travels only a short distance relative to the overall size of the network. In other words, a message exchanged between two nodes is only forwarded a maximum of $logn$ times.

Property 2.6. *The node degree (the number of edges incident to a node) in Q_k is uniformly distributed and is k [46].*

Remark 2.7. Perhaps property 2.6 is one of the weaker aspects of a hypercube. On one hand, consistent with property 2.1, all nodes have the same degree which is an advantage in network load distribution and communication protocols. On the other hand, the degree of a node increases with the size of the network. Although the increase is much slower as compared to the growth of the size of the hypercube network, as the number of nodes becomes very large so will the number of connections and the node degree. This contributes to an increased maintenance cost at each node. In addition, if the hypercube network grows to a higher dimension, nodes may need to update their connections and add additional edges. This has proven challenging for example in parallel processing where rewiring processors is a costly process. Also in distributed networks due to the absence of a central authority these global updates will involve a large number of messages. To overcome some of these difficulties a variety of constant degree hypercube variations have been designed. Some examples include Butterfly Networks, Cube Connected Cycles and Benes Networks [46].

Property 2.8. *Q_k can be constructed from two Q_{k-1} graphs. In general, a k-dimensional hypercube can recursively be constructed from lower dimensional hypercubes [46, 47].*

Remark 2.9. Property 2.8 is important in deploying and maintaining a P2P network. This property is useful to arrange the first set of nodes in a hypercube structure upon

deployment. In addition, if the number of nodes exceeds the capability of the current dimension of the hypercube we can expand to higher dimensions following a simple procedure. An example of a procedure is shown in section 2.2.2.2. Furthermore, if a large enough number of nodes leaves the network we can decrease the dimension of the hypercube allowing a more efficient use of the communication algorithms. This approach is used in some current research to build hypercube based networks. Further details are given in section 2.2.2.2.

Property 2.10. *The length of the shortest path between two nodes in a hypercube is the number of bits in which their binary labels differ. Such a number is also known as the Hamming distance. Two nodes are directly connected (neighbors) if their binary labels differ only at one position [46, 47].*

Remark 2.11. Knowledge of the path length between two nodes is an important factor in estimating the cost of sending and receiving data between any two nodes. As a consequence of Property 2.10 we can also obtain pairwise distances for any subset of nodes in a hypercube network. This is an important property for a query optimizer in a distributed database system to identify the best plan of query execution [48, 49]. We will discuss paths in more detail in section 2.2.3.1.

Property 2.12. *If the shortest path between two nodes in a hypercube is p then there are p parallel (disjoint) paths between them [47].*

Property 2.13. *Q_k has k independent (node-disjoint) and k edge-disjoint spanning trees rooted at the same node [50–52].*

Remark 2.14. Properties 2.12 and 2.13 are useful in increasing the efficiency and reliability of communication between nodes. If a node or set of nodes fails or is not available to process any requests, other alternatives can easily be identified. Knowledge of alternative paths also allows parallel processing of data. Fragments of a large data set may be sent

simultaneously using these disjoint paths thus distributing the transmission load between a larger set of nodes and minimizing the time required for such a transmission.

Property 2.15. *A hypercube structure can tolerate any number of faulty nodes not exceeding* $\lceil n/2 \rceil$ *[53] [54].*

Remark 2.16. Property 2.15 is specially important in dynamic networks where the number of nodes is changing continuously and the likelihood of node failure is relatively high.

Property 2.17. *Classes of graphs such as arrays, binary trees, meshes of trees, rings and toroids can be embedded into a hypercube optimally and efficiently [46, 47, 55].*

Remark 2.18. Property 2.17 implies that known algorithms for such structures can be extended to hypercubes. In addition, if a particular physical network resembles any of these structures, a hypercube may be used as a virtual model of communication such that nodes in the graph model represent network entities (e.g. devices, people, etc.) and edges represent virtual connections between the nodes.

Property 2.19. *A single node broadcast (i.e., a node sends the same message to all other nodes) and a single node accumulation (i.e., a node receives a response from all other nodes) require* $2^k - 1$ *messages [56].*

Property 2.20. *A multi-node broadcast (i.e., all nodes send the same message to all other nodes) and a multi-node accumulation require* $2^k(2^k - 1)$ *messages [56].*

Property 2.21. *A single node scatter (a single node sends a unique message to all other nodes) and a single node gather require* $k2^{k-1}$ *messages [56].*

Remark 2.22. Communication in a hypercube network is efficient and reliable. Properties 2.19, 2.20 and 2.21 indicate that common communication activities such as a broadcast can be performed optimally in a hypercube saving therefore valuable network resources. We discuss communication in more detail in section 2.2.3.

2.2.2 Hypercube Construction Approaches

In this section we explore a number of approaches for building and maintaining hypercube networks. We will focus on the broad concepts thus omitting minor details which may be specific to a particular implementation. Other variations exist and each particular instance of an approach will result in a different type of network and will affect some of the network properties. In many cases, a combination of different approaches is used, resulting in a hybrid network. In addition, each approach may be optimal under a given set of circumstances.

2.2.2.1 Random Network Deployment

Suppose given a set of at most $n = 2^k$ nodes, we wish to arrange these nodes as elements of a k-dimensional hypercube. There are n k-digit binary numbers. Each node is assigned one of these binary numbers at random as its unique identifier. A connection is established between two nodes if their binary identifiers differ by exactly one digit. This may require central control. A random identifier approach is used in [57] where wireless healthcare sensor nodes are connected to their virtual hypercube neighbors based on their randomly assigned binary identifier. A random identifier technique is also used in [58] to place them in a clustered hypercube DHT overlay.

A random deployment and maintenance is likely to produce a network which is misaligned, a network where two nodes that are geographically distant may be connected as neighbors given the random nature of identifier assignments. An example is given in figure 2.5. We start with a set of 8 nodes (a-h) in subfigure 2.5(a). In subfigure 2.5(b) one of the 8 possible 3-digit binary labels is assigned to each node at random. Two nodes will connect only if their binary identifiers differ by one digit. All such connections are shown in subfigure 2.5(c). Finally, in subfigure 2.5(d) nodes are arranged to show that

these connections form a 3-dimensional hypercube. Note that graphs in subfigures 2.5(c) and 2.5(d) are equivalent.

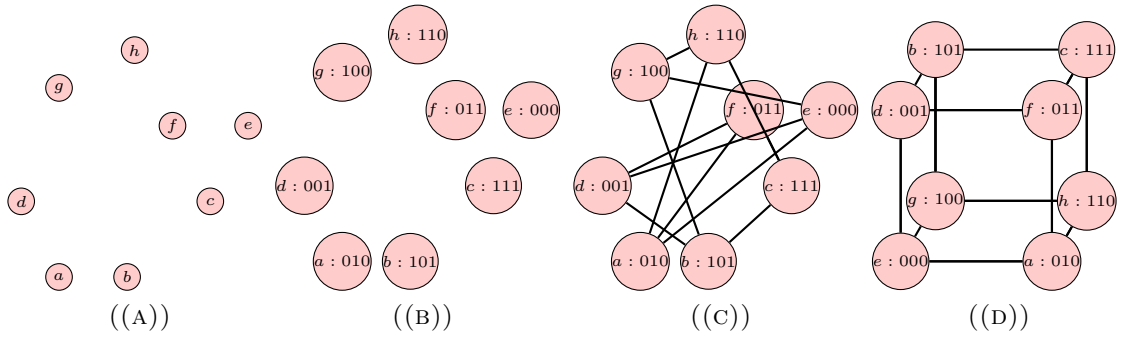

FIGURE 2.5: Example of a 3-dimensional Hypercube Construction Using Random Label Assignment

2.2.2.2 Recursive Network Deployment

A k-dimensional hypercube can be constructed from two equally labeled $(k-1)$-dimensional hypercubes. Such an operation is possible by connecting two nodes from the two lower dimension hypercubes which have the same label and pad their ids with a 0 and 1 accordingly.

The recursive property of hypercubes can also be used to expand the network to a higher dimension if the number of nodes in the network exceeds the limit of the current dimension. This can be achieved by doubling the number of labels as follows: Each label is padded with a 0 and a 1 obtaining a set of two labels. Padding is usually done at the most significant digit but it will equally work if it is done at the least significant digit. In addition, the network can recursively shrink to a lower dimension if the number of nodes decreases significantly. Figure 2.6 shows the construction of a 4-dimensional hypercube from two 3-dimensional hypercubes.

A recursive approach is used in constructing a BlueCube, a network of bluetooth connected devices [59]. Two smaller degree scatternets of bluetooth devices are combined

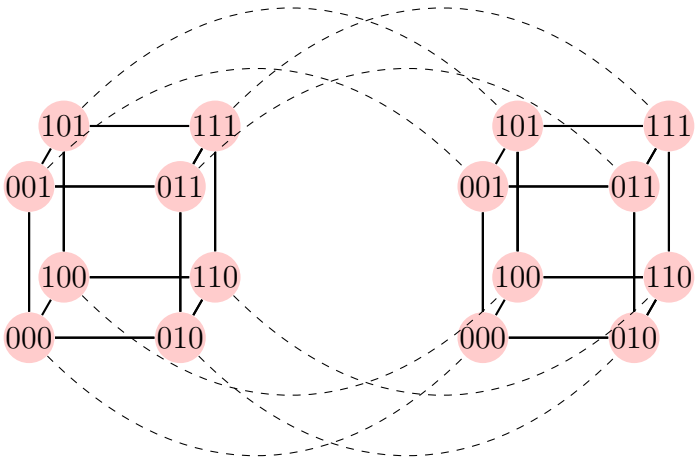

FIGURE 2.6: Example of a 4-dimensional Hypercube Construction from Two 3-dimensional Hypercubes

to form a higher dimension scatternet. In [60] a recursive greedy approach is also used where pairs of sensors with the smallest distance are first connected to form 1-dimensional hypercubes. Recursively, higher dimensional hypercubes are formed from existing lower dimension subnets. Both centralized and distributed approaches are used. Figure 2.7 illustrates the formation of a network using a greedy criteria and the recursive property of hypercubes. A set of 8 nodes is shown in subfigure 2.7(a). Four 1-dimensional hypercubes are first created as shown in subfigure 2.7(b). The choice of which pairs are selected is based on a predefined greedy criteria (e.g. node distance, link capability, etc). The four pairs are used to form two 2-dimensional hypercubes. Equal node labels are padded with a 0 or a 1 at their least significant digit. Results are shown in subfigure 2.7(c). The last step is to connect the 2-dimensional hypercubes forming a single 3-dimensional hypercube shown in subfigure 2.7(d). Finally, in subfigure 2.7(e) nodes are arranged to show that these connections form a 3-dimensional hypercube. Note that graphs in subfigures 2.7(d) and 2.7(e) are equivalent.

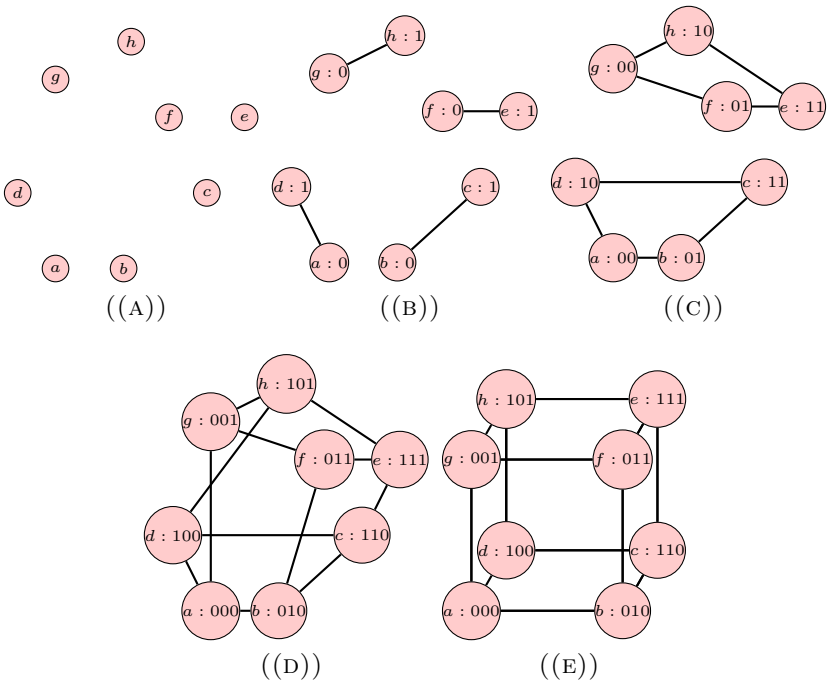

FIGURE 2.7: Example of a 3-dimensional Recursive Hypercube Construction

2.2.2.3 Dynamic Preferential Network Deployment

The initial deployment of a hypercube P2P network using a dynamic preferential attachment approach can be seen in terms of single nodes entering the network. In other words, to obtain the initial state of the network with n nodes, each one of the n nodes is sequentially added to the network. Any algorithm however can be extended to accommodate sets of nodes entering simultaneously. In addition, the same approach can be used when additional nodes enter the network during its existence.

Once a node wants to enter the hypercube network a request is sent to all existing nodes in the network. A set of nodes is selected from those which respond to the request based on a predefined criteria (fitness function). The fitness function may differ widely between networks based on their particular setting and their different applications. Finally, the new node connects to those selected new neighbors.

An example where this approach is used is HyperCup [12], a hypercube based Peer to

Peer network model. Nodes are added sequentially to the network. An implicit hypercube structure is preserved. The implicit structure is capable of supporting traditional hypercube algorithms. New nodes fill the vacant positions in the implicit hypercube. If there are no vacant positions then the hypercube expands to the next dimension.

In [60] another distributed hypercube approach is used to construct the communication tree of a hypercube. The communication tree is thereafter used to collect data from all sensor nodes and send the aggregated information to the sensor network base station. Each node during construction chooses its upstream neighbors based on their proximity. One way to satisfy the proximity criteria is to select the set of nodes which first respond to the request. A combination in this case of the geographical location of the nodes, their processing capability, traffic conditions at the moment of the request, etc. will eventually determine which nodes are able to first respond to the request. The base station however plays a role in selecting the root node in the communication tree, so the approach is not fully decentralized. In addition, if a node dies the communication tree needs to be reconstructed. We extend the idea from [60] to construct a hypercube network from a newly constructed hypercube communication tree. We achieve the above by labeling all nodes according to a hypercube broadcast labeling schema and connecting all nodes based on their assigned labels. An example is given in figure 2.8. Subfigures show the construction of a hypercube network of 8 nodes as follows; (A) the set of 8 nodes named a-h is given, (B) node 'a' is selected as the root node and labeled 000, (C) node 'a' chooses its 3 upstream nodes which are then labeled according to a hypercube broadcast tree labeling schema, (D) and (E) the hypercube broadcast tree is complete and each node has a unique label, (F) the remaining connections are added to form a full 3-dimensional hypercube shown in part (G).

This hypercube construction and maintenance approach is of particular interest for our work. HyperD which is introduced and discussed in detail in chapter 3 is a variant of a

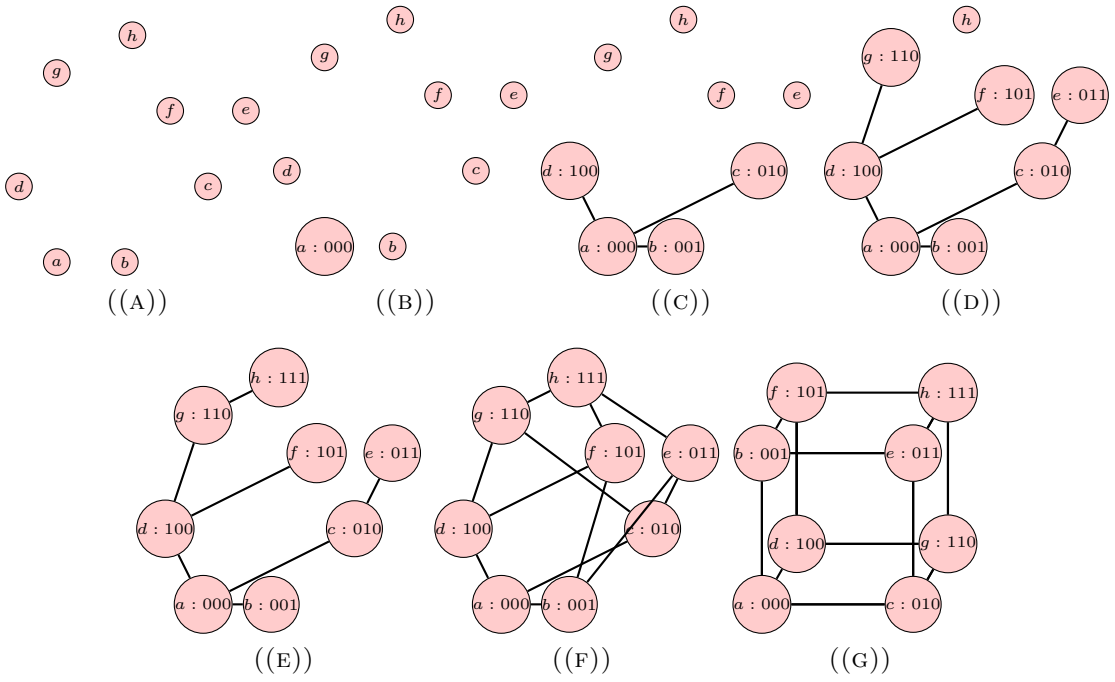

FIGURE 2.8: Example of a 3-dimensional Hypercube Communication Tree Construction

dynamic deployment based on preferential attachment.

2.2.2.4 Deployment of a Clustered Hypercube Network

In a clustered hypercube network each hypercube position is assigned to a cluster (set) of nodes as opposed to a single node. We have a hypercube of clusters instead of a hypercube of nodes. Each cluster usually communicates with nodes in other clusters via cluster ports (hubs). Nodes that belong to the same cluster share some additional information. Clusters normally have a fixed size. Once this size is reached the cluster may be partitioned into smaller clusters each occupying a different hypercube position. Clustering is a technique often used to deal with high churn and improve the security and robustness of a network. By forming clusters we limit the impact of a failure or attack to a local cluster of nodes thus preventing a systemic effect on the entire network. All issues are dealt within the cluster. Hypercubes can successfully be used to construct

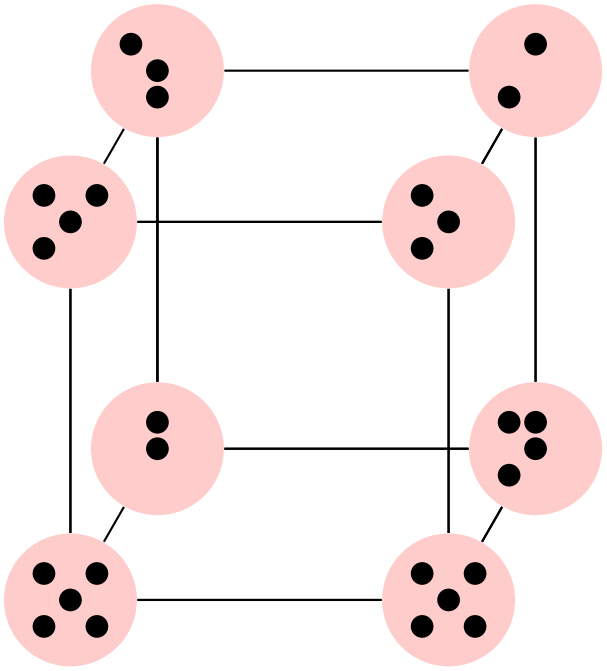

FIGURE 2.9: Examples of a Clustered Hypercube Network

a clustered system. An example is PeerCube [58], a DHT-based system where nodes are organized into clusters which form a hypercube topology. Clusters can function at different dimensions. Assignment of nodes to each cluster is done randomly based on the value of their unique identifiers. Peers are assigned to a common cluster if their identifiers share a common prefix. Another example can be found in eQuus [61] where nodes are grouped in cliques (complete graphs) sharing the same hypercube ID. All operations within the clique remain local thus increasing the resilience against high churn.

Figure 2.9 shows an example of a clustered hypercube network of 28 nodes grouped into 8 clusters. Clusters then form a 3-dimensional hypercube. Note that the network has the ability to accommodate more nodes without changing the hypercube structure. Changes usually only occur at the cluster level.

2.2.2.5 Deployment of a Hierarchical Hypercube Network

We can think of a hierarchical hypercube network as a special case of a clustered network. The difference is that each cluster is itself a hypercube. The hierarchy consists of different levels. At each level we have a hypercube structure where each position is recursively filled by smaller hypercube networks . The hypercube dimension at each level is usually the same. This approach is used to increase the expandability of the network and maintain a low node degree at any given level. Each network change is dealt with at the lower level and does not normally impact the entire network. All hypercube communication algorithms can be modified to work in such a setting. In [62] is given a summary of related work on hierarchical networks of hypercubes. In addition the Extended Hypercube (EH), a hierarchical network of parallel processors is introduced. EH is a hypercube network of smaller network elements called modules. Each module is a k-cube of processors and a Network Controller (NC). The NC is used by the module to communicate with other modules in the network. The structure can recursively be extended while still maintaining the same type of building blocks (the modules).

Figure 2.10 shows a small example similar to a network proposed in [62]. The network consists of 16 nodes. Nodes are split in groups of four. Each group (module) forms a 2-cube. The resulting set of four 2-cubes forms a higher level 2-cube of modules. Modules communicate via their getaway port (Network Controller).

2.2.3 Communication in Hypercube Topologies

Hypercube networks have been extensively studied and many optimal, efficient, distributed algorithms have been identified. In this chapter we analyze and give examples of

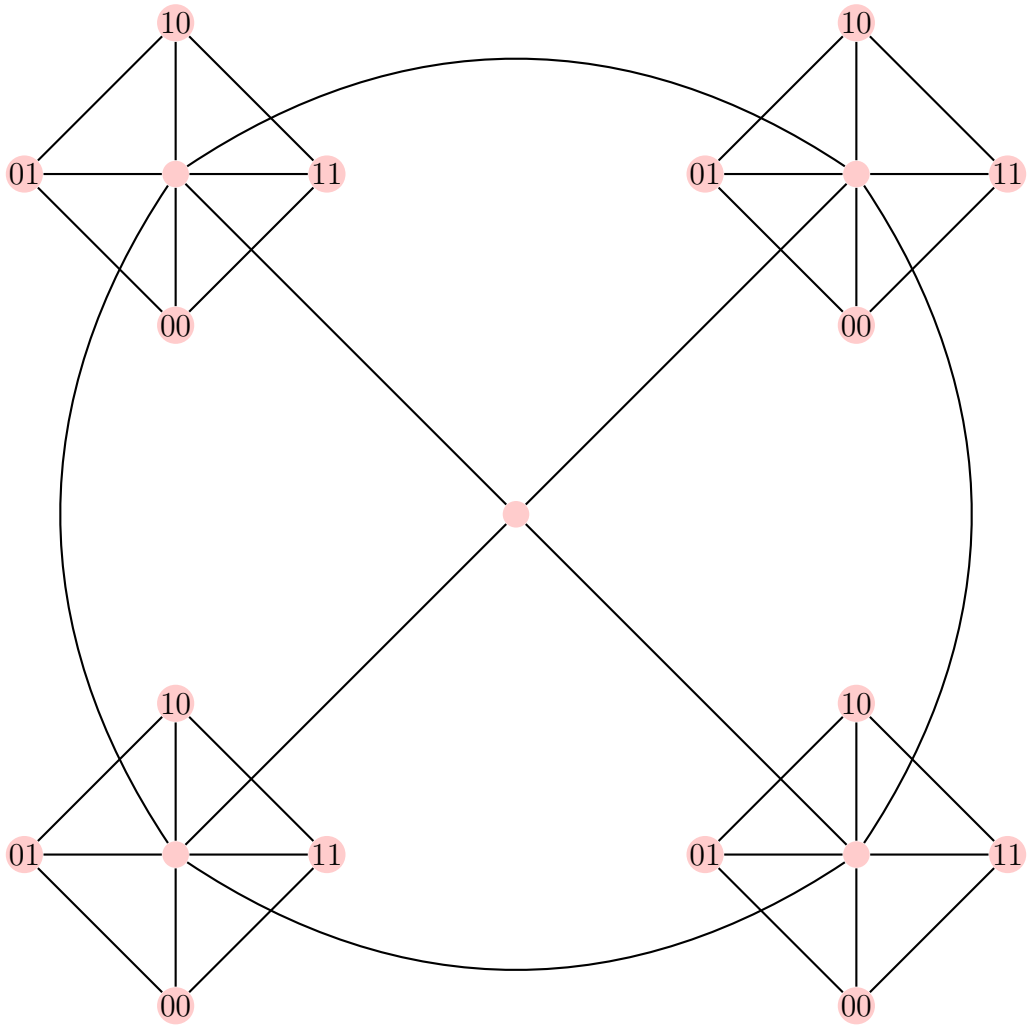

FIGURE 2.10: Examples of a Hierarchical Hypercube Network

some basic network activities such as unicasts, broadcasts and multicasts in a hypercube network.

2.2.3.1 Paths

The most common network activity is the communication between two arbitrary nodes. In a network of connected nodes many alternatives may exist for a node A to send a message to another node B. It is important that the best possible choice is selected (the shortest path is identified between the nodes). In addition, identifying the shortest path

should also be efficient. In a hypercube network a path between two nodes can easily be identified. Such a path is obtained by comparing the labels of the two nodes. The distance between the nodes is equivalent to the number of bits in which their labels differ, also known as the Hamming Distance. A message travels from node A to node B along one of the different paths between the two nodes. The sequence of hops the message has to travel is obtained by randomly selecting an intermediate node whose label is obtained by switching one of the label digits of the forwarding node. Furthermore, it is easy to identify alternative independent paths in a hypercube [47]. As noted in property 2.12 from section 2.2.1 a number of disjoint paths exist between any two nodes in a hypercube. These choices make the communication more reliable.

Figure 2.11 shows an example of two disjoint paths between two nodes, 001010 and 101101 in a 6-dimensional hypercube. Since the two labels differ at 4 positions, the shortest path between them is 4 hops. Two of the paths are shown. Note that any two of the adjacent nodes in a path differ in only one bit position. In addition, there are no common nodes between the two paths. The failure of one or more nodes in the first path does not affect communication in the alternative path.

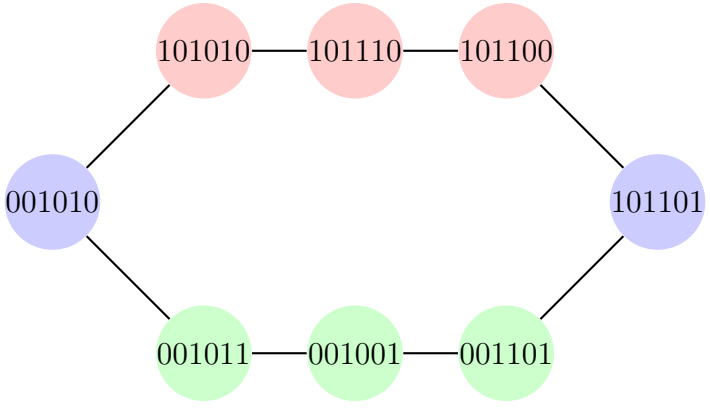

FIGURE 2.11: Independent Parallel Paths

2.2.3.2 Broadcast

A broadcast is one of the most common operations in a network of peers. We can differentiate between a single node broadcast where a message is sent from one node to all the other nodes and a multi-node broadcast where all nodes send a message to all other nodes. In a hypercube network a single node broadcast can be efficiently implemented by using one of the many possible minimum spanning trees [51, 56].

In Figure 2.12 an example of a spanning binomial tree [51] in a 4-dimensional hypercube is given. The example follows a simple approach:

1. The broadcasting node 0000 sends the message to all its neighbors.
2. If a node v receives a message from its i-dimensional neighbor u (two nodes are i-dimensional neighbors if their labels differ exactly at the i^{th} position), it forwards the message to all of its k-neighbors where $0 < k < i$.

The example in figure 2.12 however, only works in those cases when the nodes in the network form a complete hypercube. As discussed earlier in this chapter, very often the hypercube is not complete, some positions may be vacant or some of the nodes manage more than one hypercube position forming an implicit complete hypercube. Figure 3.5 shows an example taken from [63] showing a broadcast in an incomplete 4-dimensional hypercube.

The broadcast tree with root at node (5) is given. The broadcast in this partial hypercube 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 happen that a node receives a message more than once. For example in figure 3.5 node (10) receives the message twice. In general, under normal

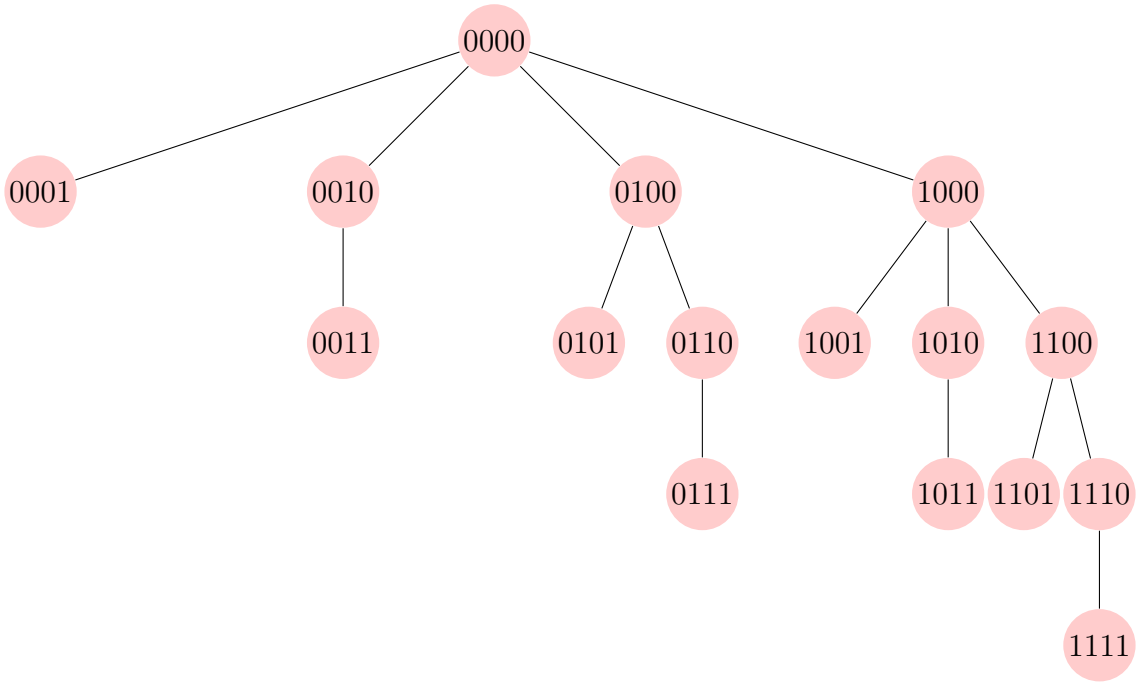

FIGURE 2.12: A Spanning Binomial Tree in a 4-dimensional Hypercube

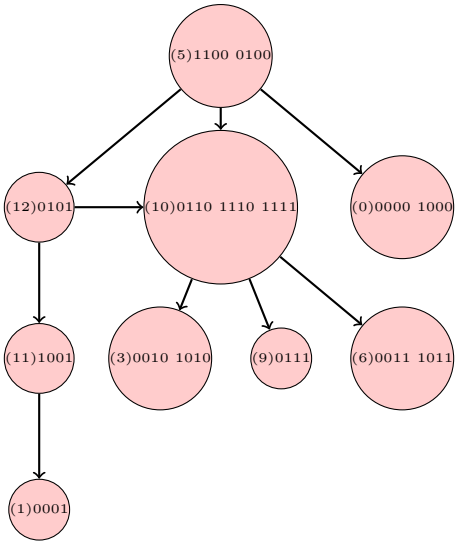

FIGURE 2.13: Example of a Broadcast in a Partial 4-dimensional Hypercube Network

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

In a dynamic network, frequent changes may disrupt the normal flow of communication between nodes. In section 2.2.1 we discussed some of the aspects that make hypercubes

a good choice in minimizing the impact of node failures. When broadcasting a message along a single spanning tree it may happen that the failure of one or more nodes may disrupt further propagation of the broadcast. One way to increase the probability that all available nodes receive the message is to send it along several independent spanning trees. It has been shown that by doing so we can tolerate a number of faulty nodes up to $n/2$. Figure 2.14 from [52] shows an example of 3 independent spanning trees in a 3-dimensional hypercube.

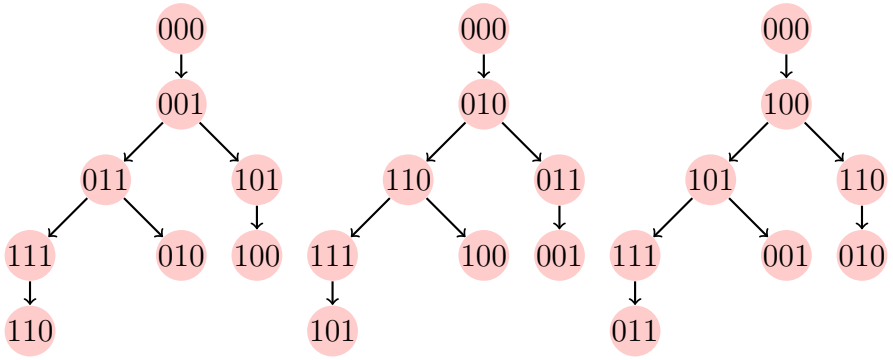

FIGURE 2.14: Example of Edge-disjoint Spanning Trees for a 3-dimensional Hypercube

2.2.3.3 Multicast

We define the multicast as a message sent from one node to a subset of k nodes in the network. A simple way to do so is to send k copies of the message separately. This only works well when the destination nodes are near the source node. An optimal solution in terms of network traffic implies finding the communication tree (multicast tree) with the minimum cost or the communication path (multicast path) with the minimum cost. Such problems are known to be NP-Hard [64]. Several different multicasting tree versions such as the minimum steiner tree [64] are known. A variety of communication path versions such as the shortest path multicast [64] are also known. In a hypercube network however it is possible to find heuristics to significantly reduce complexity with only a small increase

in network traffic. Examples of heuristic multicast algorithms on hypercubes as well as a comparison of the performance of other contemporary algorithms can be found in [65].

Chapter 3

HyperD: A Distributed Hypercube Network

3.1 Introduction and Background

HyperD is a hypercube overlay for fully distributed P2P networks. All nodes belonging to HyperD are organized in a virtual complete hypercube. HyperD is scalable and fault tolerant. The underlying physical network however is unlikely to resemble a full hypercube either in the way that nodes connect or in the number of nodes required to fill all 2^k possible positions belonging to a full k-dimensional hypercube. To deal with any missing hypercube positions, existing neighboring nodes are allowed to gain temporary control of vacant positions. The structure is still able to make efficient use of known communication algorithms which take advantage of the hypercube structure.

We first introduced HyperD in two papers [63, 66]. HyperD was initially designed to serve as a topology model for a Dynamic Distributed Federated Database (DDFD) [67]. A DDFD is a fully distributed relational database system which is well suited to a large

range of database applications where data is stored locally at each node but is available and accessible to all other nodes in the network. A DDFD is indeed an example of a P2P network, so all results shown in these papers apply to more general cases of P2P networks. Special interest was given to a particular DDFD implementation, namely, the Gaian Database (GaianDB) [67]. GaianDB grows by a process of preferential attachment. The resulting overlay network features properties of a scale free graph. This is a typical example of an unstructured P2P overlay network where flooding is a common method used to search for content and identify communication routes. The advantages of using the GaianDB in improving query processing as well as the behavior and properties of such networks have been shown in [68–70]. We then explored further methods for improving performance in GaianDB networks. In [48, 49] we argued that using a fixed graph structure such as the hypercube offers a number of advantages in this type of distributed environment. In order to verify these positive theoretical predictions we decided to come up with a practical implementation of a hypercube network. These were the first steps in designing HyperD, which adopts a growth model that is well suited to the type of dynamic distributed environment similar to the GaianDB system.

Our approach differs from most previous work in that the integrity of the hypercube is maintained by allowing nodes to take control of multiple hypercube positions. At any given time labels representing these hypercube positions are shared among nodes and redistributed when network changes occur. In addition, we employ a method of preferential attachment to assign labels to new arriving nodes so that virtual neighbors are also relatively close in the underlying physical network. Maintaining the integrity of the hypercube allows us to adopt, with some modifications, traditional hypercube communication algorithms.

3.1.1 Other Related Work

Hypercubes have been extensively studied and used in a variety of applications. In some early work hypercubes were commonly used in parallel processing. Given the popularity of parallel computation using structured topologies such as the hypercube, a large number of research papers and books have been written on the subject. A detailed survey of the use of hypercubes and other known structures in parallel processing, including an extensive bibliography can be found in [55]. A more recent example of the use of hypercubes in parallel processing can be found in [71]. In [71] a constant degree hypercube variant is used to construct a topology which is scalable while maintaining positive hypercube properties such as low diameter and efficient routing. Another related use of hypercubes can be found in [72] where a hyper-graph partitioning model is used to minimize communication cost in dynamic scientific computations.

Hypercubes have also been applied in wireless and sensor networks. In [59] a hypercube communication network for bluetooth devices is proposed. The resulting fault tolerant communication scheme takes advantage of known hypercube communication algorithms to reduce the network workload thus efficiently using network resources such as bandwidth and battery power. In [60] a data gathering scheme is described for sensor networks based on a hypercube topology. Sensor nodes which have limited energy resources benefit from the hypercube arrangement which reduces transmission times, distributes the load over the network, therefore increasing the system resilience against failure. Another example of the use in wireless communication can be found in [57]. An efficient hypercube based wireless healthcare network is presented. The authors show that the proposed network works well in both static and dynamic environments by reducing latency and communication overhead.

In a P2P environment a hypercube can also be used as a model for implementing an

overlay network based on a Distributed Hash Table (DHT) [73]. Examples in [58] and [61] use hypercube overlay structures to maintain a DHT which greatly improves information storage, search and retrieval.

3.2 HyperD

3.2.1 Model and Preliminaries

The following notation and definitions are used in this chapter.

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 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 $e \leq n2^{n-1}$.

Definition 3.1 (Full and Partial Hypercube)**.** D is a *full hypercube* if it has 2^k nodes, each owning a single label thus occupying a unique position in the k-dimensional hypercube. Therefore if D is a *full hypercube*, then $V_D = V_H$ and $E_D = E_H$. Otherwise D is a *partial hypercube*.

Definition 3.2 (Label Space)**.** 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. The *Label Space* $LS(v)$ of a node $v \in D$ is the set of binary labels assigned to v. Since a node in a partial hypercube D may have to manage several labels to allow for the preservation of an implicit complete hypercube, $LS(D) = LS(H)$.

HyperD is a hypercube P2P overlay model featuring the following properties:

- HyperD is a set of interconnected peers which organize and communicate in a purely ad-hoc fashion without help from any centralized authority.

- Nodes in HyperD establish bidirectional connections which correspond to edges in the graph D modeling the network. Thus D is an undirected graph.
- HyperD is dynamic and nodes may enter or leave the network at any time. Label assignments and connections in a HyperD are rearranged to deal with these changes.
- HyperD approximates an n-dimensional hypercube if the number of nodes is between 2^{n-1} and 2^n. Operating at a higher hypercube dimension however is unlikely to impact the performance of the network.
- Each node $v \in$ 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 $v \in$ HyperD knows the labels for each of its neighbors.
- If the number of nodes exceeds 2^n, HyperD expands to a virtual $(n+1)$-dimensional hypercube.

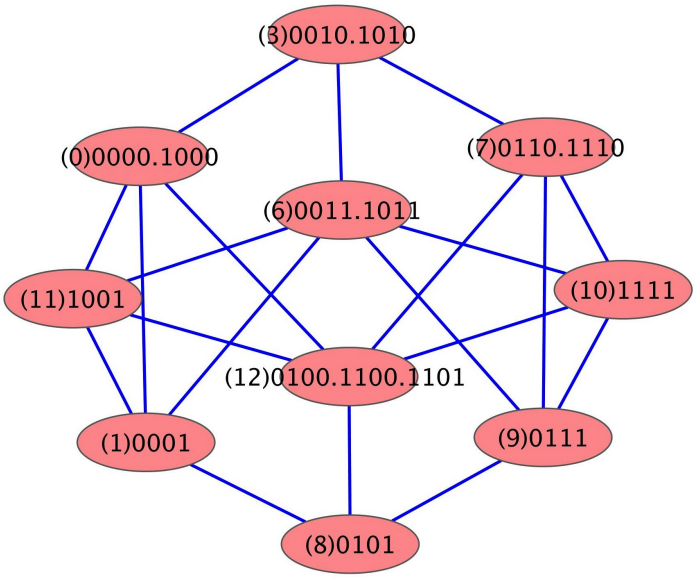

FIGURE 3.1: Example of a 4-dimensional HyperD of 10 Nodes

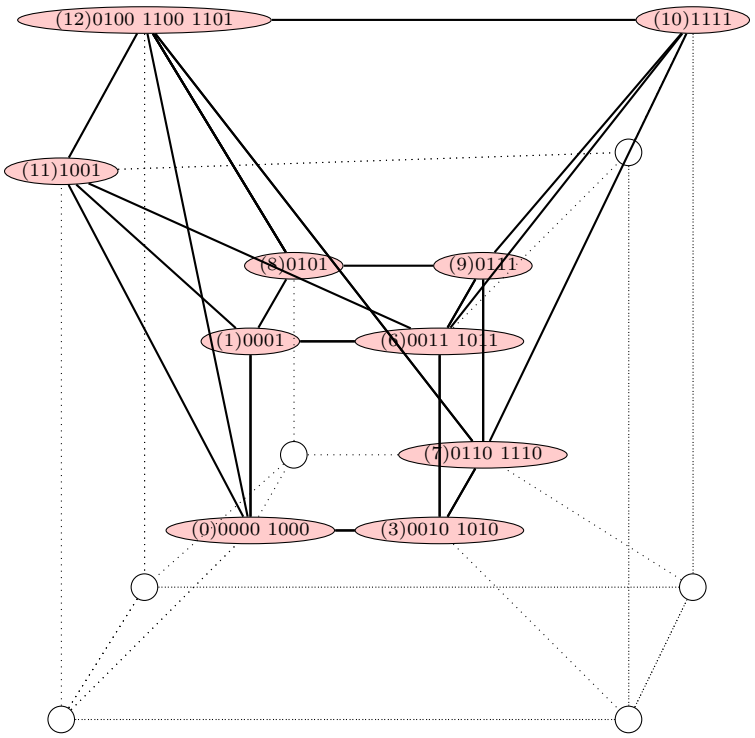

FIGURE 3.2: HyperD Network from Figure 3.1 with Nodes Arranged on a Hypercube Frame

Figure 3.1 shows an example of a HyperD of 10 nodes. During the existence of this network 13 nodes have entered, 3 of which have at some point left the network. The remaining 10 nodes form a partial (incomplete) 4-dimensional HyperD. In Figure 3.2 we show the same HyperD network with nodes arranged on a 4-dimensional hypercube frame to illustrate the position within the hypercube structure occupied by each node. $(u)LS(u)$ is a node u with label space $LS(u)$. ◯ indicates a vacant position in the underlying hypercube structure. A solid line denotes an edge joining two nodes in HyperD, and a dotted line represents a vacant edge in the underlying hypercube.

3.2.2 Communication in HyperD

In a P2P network, available resources such as bandwidth and processing power may be limited. To reduce the network load and increase throughput it is important to reduce

the cost of communication between nodes. Broadcasting and pairwise communication are the most frequent activities in a distributed network. In this section we adopt known hypercube communication algorithms to implement routing and broadcast procedures.

3.2.2.1 Routing

First we present a routing algorithm for HyperD. A path between two nodes is determined by their respective label sets and those of any intermediate nodes . Details are summarized in Algorithm 1. Some of the definitions used in this algorithm are listed below:

Definition 3.3 (Neighborhood)**.** 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 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 $LS(v)$. Any node $u \in V_D$ has a set of neighboring nodes denoted by $neighborhood(u)$.

Definition 3.4 (Edges)**.** Let $u \in V_D$ and $v \in V_D$ be two neighboring nodes. Then $edge(u, v)$ indicates that a virtual connection exists between the two nodes which is represented by an undirected edge in the HyperD graph. As noted earlier, two nodes that share an edge may not necessarily be neighbors in the underlying physical network. They do, however, maintain the necessary information to allow communication between them.

Definition 3.5 (Node Distance)**.** Let $difference(l_i, l_j)$ be the bit difference between two labels $l_i \in LS(D)$ and $l_j \in LS(D)$. Then, for any two nodes $u \in V_D$ and $v \in V_D$, $distance(u, v) = min(difference(l_u, l_v))$ where $l_u \in LS(u)$ and $l_v \in LS(v)$.

Algorithm 1 recursively identifies a path connecting the source node and destination node. The base case (Steps 2-3): if the destination node is a neighbor of the source node than the message is delivered directly and routing ends. Otherwise the message is

Algorithm 1 HyperDRouting(D, s, d)

Input: HyperD Network D, Source Node $s \in V_D$, Destination Node $d \in V_D$.
Output: A path P from s to d, where $P \subseteq E_D$.

1: $P \Leftarrow \emptyset$
2: **if** $d \in neighborhood(s)$ **then**
3: $\quad P \Leftarrow P \cup \{edge(s,d)\}$
4: **else**
5: $\quad$ Select a node $t \in neighborhood(s)$ such that $min(distance(t,d))$. Ties are broken randomly.
6: $\quad P \Leftarrow P \cup \{edge(s,d) \ \cup HyperDRouting(D,t,d)$
7: **end if**
8: **return** P

forwarded to a neighbor whose distance to the destination node is the smallest (Steps 4-6). At each intermediate node along the path the same criteria is used to forward the message. Furthermore, each node will usually have multiple choices of where to forward the message, since multiple paths between any two nodes exist in a hypercube. The availability of such choices is crucial in a dynamic network where nodes may fail and thus may eliminate a particular path between two nodes.

Proposition 3.6. *Let $D = (V_D, E_D)$ be a graph representing a HyperD instance with $2^{n-1} < |V_D| \leq 2^n$.Let$H$=(V_H, E_H) be a n-dimensional hypercube with 2^n vertices serving as a model for D. Let $route(a,b)$ be the number of links from D used to send a message from node $a \in V_D$ to node $b \in V_D$. Then we have the following:*

1. *$route(a,b) \leq n$ for every node $a, b \in V_D$.*
2. *If D is full (D is equivalent to H), then $route(a,b)$ is optimal and is at most $log_2(|V_D|)$.*
3. *The characteristic path length (CPL) for D is at most $\frac{log_2(V_D)+1}{2}$.*

Proof. D by definition is expected to preserve the integrity of the hypercube structure used as a model. So each vertex and edge from H is explicitly or implicitly preserved in

D. Thus (1) is a direct consequence of property 2.10. Similarly, if D is full, than there is a one-to-one correspondence between the nodes (and edges) of D and H. Thus (2) also is a direct consequence of property 2.10. Furthermore, it is well known that CPL in a n-dimensional hypercube is $n/2$. As a consequence the CPL in H is $log_2(V_H)$. Assume $2^{n-1} < |V_D| \leq 2^n$. When $V_D = 2^{n-1}$, D functions as a full $(n-1)$-dimensional hypercube. Upon expansion to a n-dimensional hypercube, the length of each label in D is increased by 1. Thus the path between any two nodes in D would increase by at most one unit. In conclusion, CPL for D is at most $\frac{log_2(V_D)+1}{2}$. This upper bound is not tight given the fact that some of the hypercube edges are likely to be internal to a node in D. □

Remark 3.7. If HyperD is partial, with one or more nodes occupying multiple hypercube positions, then the actual shortest path length between two of its nodes u and v may be less than $distance(u, v)$ since a single node may own two or more labels belonging to the same path. A direct consequence is the fact that the path chosen may not necessarily be optimal. The difference between a chosen path and an optimal path is expected to be very small.

Figure 3.3 shows two possible paths from node (001010) to node (101101). Since every hop is chosen based on each neighbor label distance to the destination node, the longer path has the same chance of being selected as the shorter path.

3.2.2.2 Broadcast

It is quite common for a node to send a message (query) to all other nodes in the network. Optimal broadcast algorithms exist for n-dimensional hypercubes $H = (V_H, E_H)$ 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 [56].

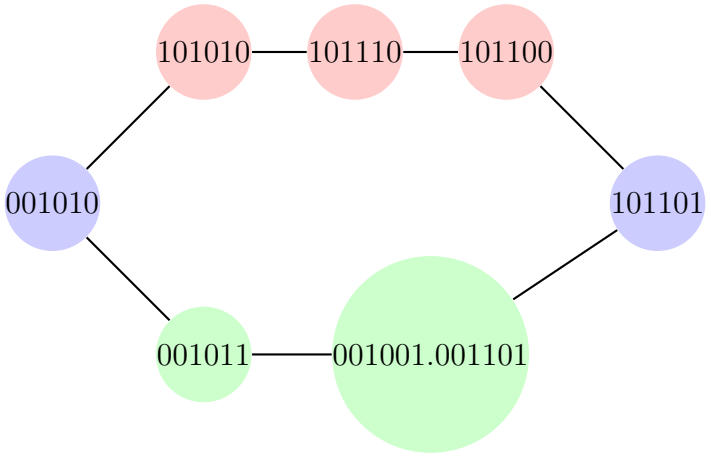

FIGURE 3.3: Comparison of Two Different Paths in a Partial HyperD

Definition 3.8 (i^{th} neighbors)**.** Two distinct nodes in a full hypercube are said to be *i neighbors*, $0 \leq i < n$, if their binary labels differ only in the ith position. Let nodes $u \in V_D$ and $v \in V_D$ be *neighbors*. Let i be the smallest value of the bit position for which any label $l_u \in LS(u)$ is adjacent to any label $l_v \in LS(v)$. Then u is $neighbor_i(v)$ and v is $neighbor_i(u)$. We also denote the bit position in which two adjacent labels l_u and l_v differ as $dimension(l_u, l_v)$. In the current context, $dimension(l_u, l_v) = i$. As a special case $dimension(l_u, l_u) = -1$.

A simple recursive broadcasting procedure in a hypercube is described in Algorithm 2.

Algorithm 2 can be summarized as follows: The broadcasting node sends the message to all its neighbors (Steps 4-7). In turn, each node v upon receiving a message from its i-dimensional neighbor u, forwards the message to all of its k-neighbors where $i < k < n$ (Steps 8-12). The algorithm returns a spanning tree with root at the source node (Step 13). Algorithm 2 requires exactly $n-1$ messages to reach all $n-1$ nodes in a hypercube not including the source node (see property 2.19 in chapter 2 and the reference therein).

An example where Algorithm 2 is used to broadcast a message in a 4-dimensional hypercube originated at node 

is illustrated in figure C.5.

Algorithm 2 HypercubeBroadcast(H, s, c)

Input: HyperD D, Source Node $s \in V_H$, Current Node $c \in V_H$.
Output: A broadcast (spanning) tree B, where $B \subseteq E_H$.

$B \Leftarrow \emptyset$
$k \Leftarrow$ dimension of H
$dim \Leftarrow$ dimension of neighbor s
if $s = c$ **then**
  for $i = k - 1$ to 0 **do**
    $B \Leftarrow B \cup \{edge(c, neighbor_i(c))\} \cup HypercubeBroadcast(H, c, neighbor_i(c))$
  end for
else
  for $j = k - 1$ to $dim + 1$ **do**
    $B \Leftarrow B \cup \{edge(c, neighbor_j(c))\} \cup HypercubeBroadcast(H, c, neighbor_j(c))$
  end for
end if
return B

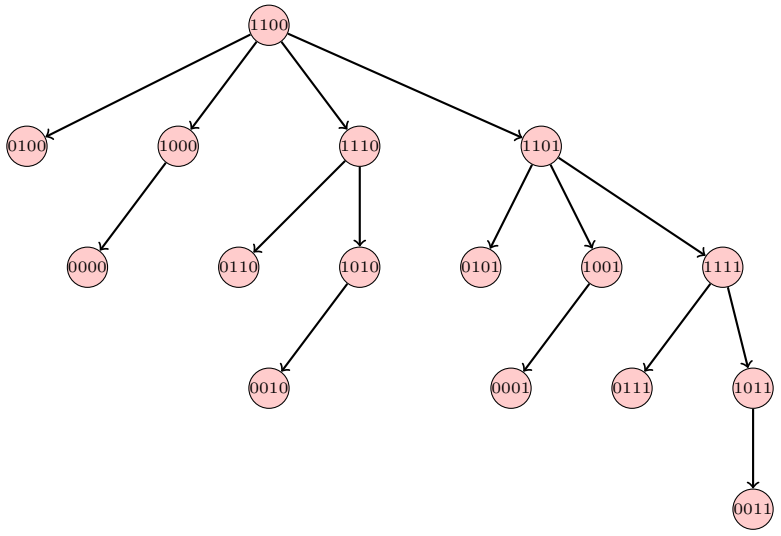

FIGURE 3.4: Example of a Broadcast in a 4-dimensional Hypercube

We adopt this approach to fit the specific characteristics of HyperD. Algorithm 3 shows the steps used to broadcast a message in a HyperD network. We have to consider the fact that each node in HyperD may own more than one hypercube label.

Algorithm 3 is used to identify one of the several possible spanning trees with root at the broadcasting node. We offer additional comments here for a better understanding of the algorithm. The broadcasting node s selects a label $l_s \in LS(s)$ and sends the

Algorithm 3 HyperDBroadcast(D, s, c, l_s, l_c)

Input: HyperD Network D, Source Node $s \in V_D$, Current Node $c \in V_D$, Label $l_s \in s$, Label $l_c \in c$.
Output: A broadcast (spanning) tree B, where $B \subseteq E_D$.

1: $B \Leftarrow \emptyset$
2: $dim \Leftarrow dimension(l_s, l_c)$
3: $k \Leftarrow$ dimension of D
4: **for** each node $N_i \in neighborhood(c) \neq s$ **do**
5: **if** $difference(l_c, l_i \in N_i) == 1$ && $dimension(l_c, l_i) > dim$ **then**
6: $B \Leftarrow B \cup \{edge(c, N_i)\} \cup HyperDBroadcast(D, c, N_i, l_c, l_i)$
7: **end if**
8: **end for**
9: **for** each label $l_i \in c$ **do**
10: **if** $difference(l_c, l_i) == 1$ **then**
11: $B \Leftarrow B \cup HyperDBroadcast(D, c, c, l_c, l_i)$
12: **end if**
13: **end for**
14: **return** B

message to all its neighbors relative to l_s (i.e., to those neighbors having labels adjacent to l_s). Each message includes the chosen label l_s. The broadcasting node would invoke $HyperDBroadcast(D, s, s, l_s, l_s)$ to initiate the broadcast. Recursively, if a node c receives a message from its neighbor s (including label l_s), it forwards the message to all its $neighbor_k(c)$ $(i < k < n)$ including label l_c. (Steps 4-8). Note that a node may have more than one label adjacent to l_u. If c itself has a label l_w that is a j-neighbor of l_c where $j > k$, it sends a message to all its neighbors having a label that is a $g - neighbor$ of l_w $(j < g < n)$ including label l_w (Steps 9-13).

Figure 3.5 shows an example of a broadcast in a partial 4-dimensional HyperD with 10 nodes (from figure 3.2).

Figure 3.6 shows another computer generated example of a broadcast in a partial 5-dimensional hypercube of 25 nodes. The broadcast is initiated at node (6). Node size and color gradient are determined using in-degree information. A larger example can be found in Figure B.12 from Appendix B.

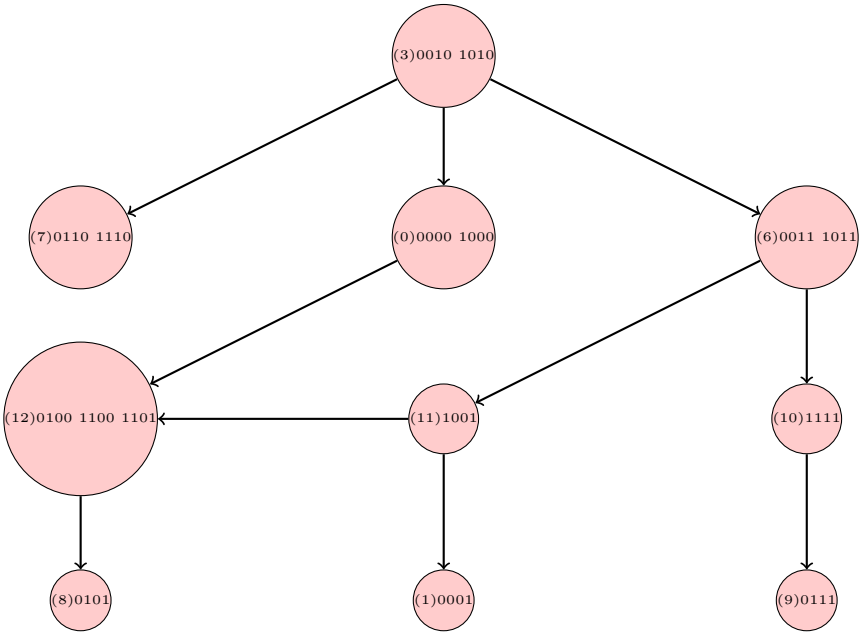

FIGURE 3.5: Example of a Broadcast in a 4-dimensional HyperD

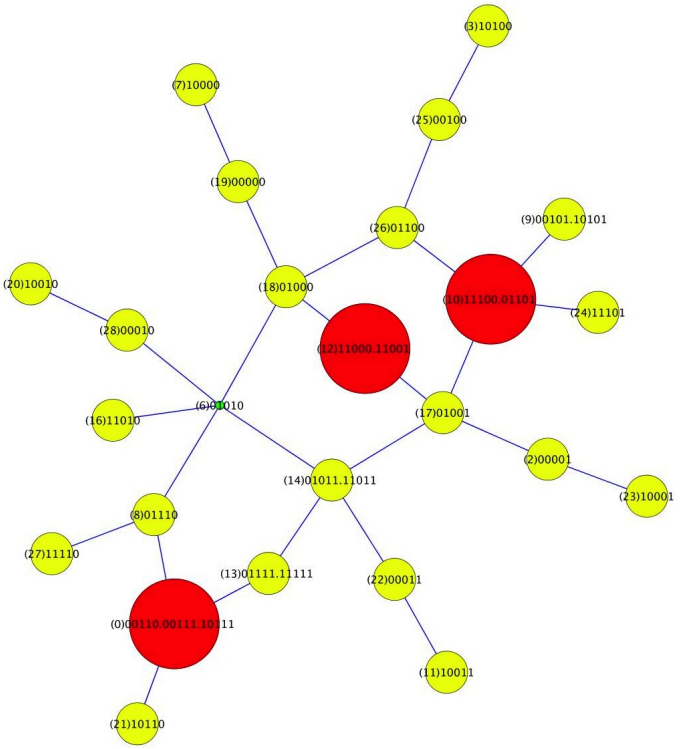

FIGURE 3.6: Example of a Broadcast on a 5-dimensional HyperD of 25 Nodes. Node Size and Color Gradient Determined by Node In-degree

Remark 3.9. If we have a full HyperD then the results produced by Algorithms 2 and 3 are identical. In a partial HyperD of n nodes more than $n-1$ messages may be necessary to reach all nodes. This is a consequence of a single node managing two or more hypercube positions. This may cause a small number of redundant messages. For example, in Figure 3.6 node size is determined by the number of incoming messages. It can be clearly seen that three nodes each received a message from two distinct nodes during the broadcast.

3.2.3 HyperD Deployment

We can think of the initial deployment of a HyperD instance as the process of adding the nodes sequentially to the new virtual hypercube overlay. The steps followed by a node entering the network and the label assignment procedure are the same for each node regardless of the node belonging to the initial set of nodes upon deployment or if we are dealing with a new node entering HyperD at any point during the existence of the network.

3.2.4 HyperD Maintenance

Network maintenance is the process of dealing with any network changes such as nodes entering the network, nodes leaving or nodes moving within the network. We can think of a node moving within the network as a process involving two steps, the node leaving the current position and the node joining a different part of the network. Therefore our analysis will only focus on determining the necessary steps to deal with node enters and node deletions.

3.2.4.1 Node Enters

When a new node u intends to enter a HyperD network it first needs to contact some existing node $n_0 \in$ HyperD. The principal goal is to identify a donor node which has an available label. In the best case scenario, n_0 may itself be able to share a label with u. However, chances are that n_0 has a single label. In that case 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 $|LS(n_i)| > 1$ (i.e., if 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 the enter request to any other nodes. This limits the broadcast range, thus saving valuable network resources. The new node u is very likely to receive multiple label offers. Node u would then choose its new label based on a predefined criteria (such as node proximity, or donor node label size). Such criteria can be further refined based on the scope and need of a specific network. Algorithm 4 is used for an enter broadcast.

Algorithm 4 HyperDEnterBroadcast(D, s, c, l_s, l_c)

Input: HyperD Network D, Source Node $s \in V_D$, Current Node $c \in V_D$, Label $l_s \in s$, Label $l_c \in c$.
Output: Set $A \subseteq V_D$ of nodes with more than one label.

$A \Leftarrow \emptyset$
$dim \Leftarrow dimension(l_s, l_c)$
$k \Leftarrow$ dimension of D
if $|LS(c)| > 1$ **then**
  $A \Leftarrow A \cup \{c\}$
else
  for each node $N_i \in neighborhood(c)$ **do**
    if $difference(l_c, l_i \in N_i) == 1$ && $dimension(l_c, l_i) > dim$ **then**
      $A \Leftarrow A \cup HyperDEnterBroadcast(D, c, N_i, l_c, l_i)$
    end if
  end for
end if
return A

Algorithm 4 can be summarized as follows: If the broadcasting node has itself available labels there is no need to forward the request and the broadcast will end (Steps 2-5). Otherwise the node will forward the message to all its neighbors which recursively will repeat the same process (Steps 6-12). The algorithm will return to the broadcasting node a set of nodes A which have available labels. If HyperD is a *partial hypercube* the new node will receive at least one label offer, $|A| \geq 1$. Otherwise if HyperD is a full hypercube (no available labels) the new node receives no labels, therefore $|A| = 0$. In the event $|A| = 0$, HyperD has to expand to the next dimension to allow the new node to enter. Each node in the network will have to double its label space by padding each label with a 1 and 0 at its highest dimension to create this way two distinct labels.

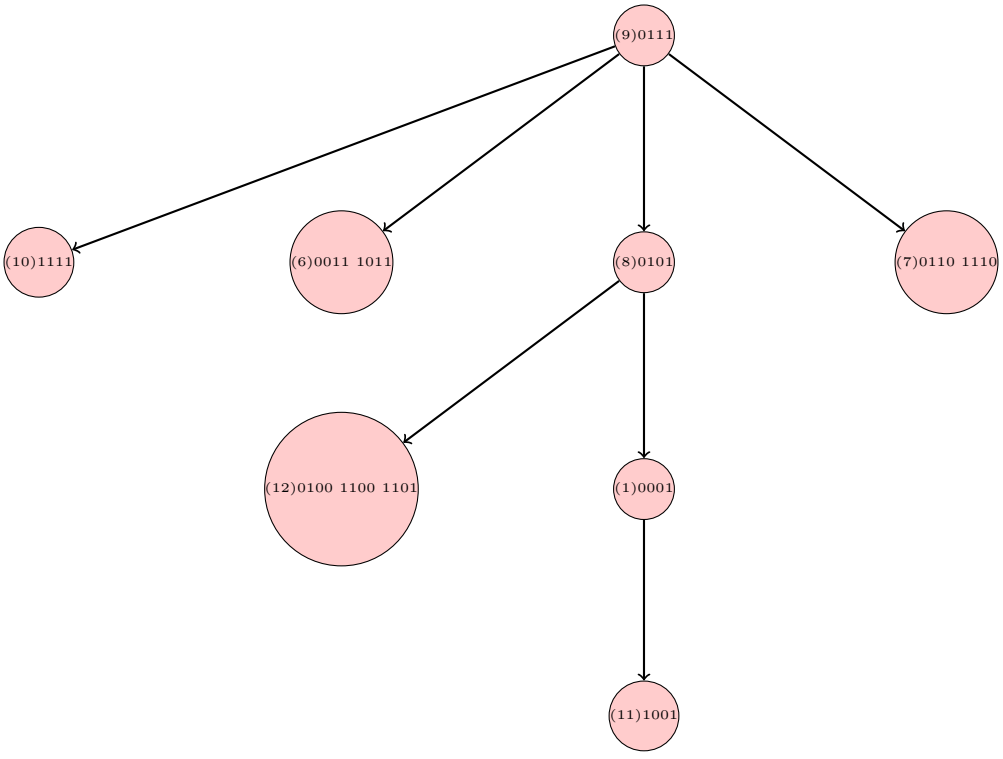

FIGURE 3.7: Example of a Enter Broadcast in a 4-D HyperD

Figure 3.7 illustrates Algorithm 4. We use the HyperD network shown in figure 3.2. The broadcast originates at node 

. Note that only 8 of the 10 nodes are reached by this broadcast.

Algorithm 5 HyperDEnter(D, n, v)

Input: HyperD D, New node $n \notin V_D$, Node $v \in V_D$.
Output: A new HyperD D including the new node n.

1: //Node n contacts node $v \in V_D$
2: $L \Leftarrow HyperDEnterBroadcast(D, v, v, l_v, l_v)$
3: **if** $L = \emptyset$ **then**
4: Expand D to the next dimension
5: $L \Leftarrow HyperDEnterBroadcast(D, v, v, l_v, l_v)$
6: **end if**
7: Node $a \Leftarrow Null$
8: **for** each node $i \in L$ **do**
9: **if** $distance(i, v) < distance(a, v)$ **then**
10: $a \Leftarrow i$
11: **end if**
12: **end for**
13: Node a assigns a label to n
14: **for** each node $k \in neighborhood(a)$ **do**
15: **if** $distance(n, k) == 1$ **then**
16: $E_D \Leftarrow E_D \cup edge(n, k)$
17: **end if**
18: **if** $distance(a, k) > 1$ **then**
19: $E_D \Leftarrow E_D - edge(a, k)$
20: **end if**
21: **end for**
22: **return** D

The *new node enter broadcast* is the most important step in the enter procedure which is shown in Algorithm 5. The following definition is needed here.

Definition 3.10 (Internal Edges)**.** 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.

The new node n identifies a node v from the HyperD network (Step 1). Node v then broadcasts the enter request. A list of nodes with available labels returned by the broadcast are stored in L (Steps 2-6). The new node then selects a node a from L which is the closest in the virtual network as its first neighbor. (Steps 7-12). Choosing the closest node helps maintain a better alignment between the virtual network and the physical

network. Node a then donates one of its labels to the new node n (Step 13). Node a chooses the label to donate using the following criteria: a) the label with the smallest internal degree; b) ties are broken by selecting the label with the largest binary value. Upon receiving the label the new node connects to its new neighbors (Steps 15-17). The donor node a disconnects from those neighbors related to the donated label (Steps 18-20).

Proposition 3.11. *Let $D = (V_D, E_D)$ be a graph representing a HyperD instance with $2^{n-1} < |V_D| \leq 2^n$. Let $H = (V_H, E_H)$ be a n-dimensional hypercube with 2^n vertices serving as a model for D. Let $enter(a)$ be the number of messages sent over the links in D to deal with the addition of node a. Then $enter(a)$ is in $O(V_D)$.*

Proof. Assume node a is entering a full HyperD. Then the broadcast issued to identify a new label would unsuccessfully reach all current nodes using $V_D - 1$ messages. Subsequently, a new broadcast is issued to inform all nodes to expand to the next hypercube dimension. This is the worst case scenario since in all other instances the broadcast does not necessarily need to reach all nodes in the network and does not need to expand to a higher dimension. $enter(a)$ would therefore require at most two broadcast messages in the worst case scenario. Once the label has been identified the new node needs to connect to at most $log_2 V_D$ neighbors, each no more than $log_2 V_D$ steps away. Clearly the cost in terms of messages for identifying a new label outweighs the cost of connecting to all new neighbors. Thus $enter(a)$ is in $O(V_D)$. □

Figure 3.8 shows an example of a node entering a full 4-dimensional HyperD. Node (16) broadcasts an enter request. Since the network is a full hypercube and no labels are available, it needs to expand to a 5-dimensional HyperD. Node (15) which was the first to be contacted by the new entering node, donates its largest binary label 00101 to (16). Upon securing the new label, node (16) connects to nodes (1), (6), (7), (9) and

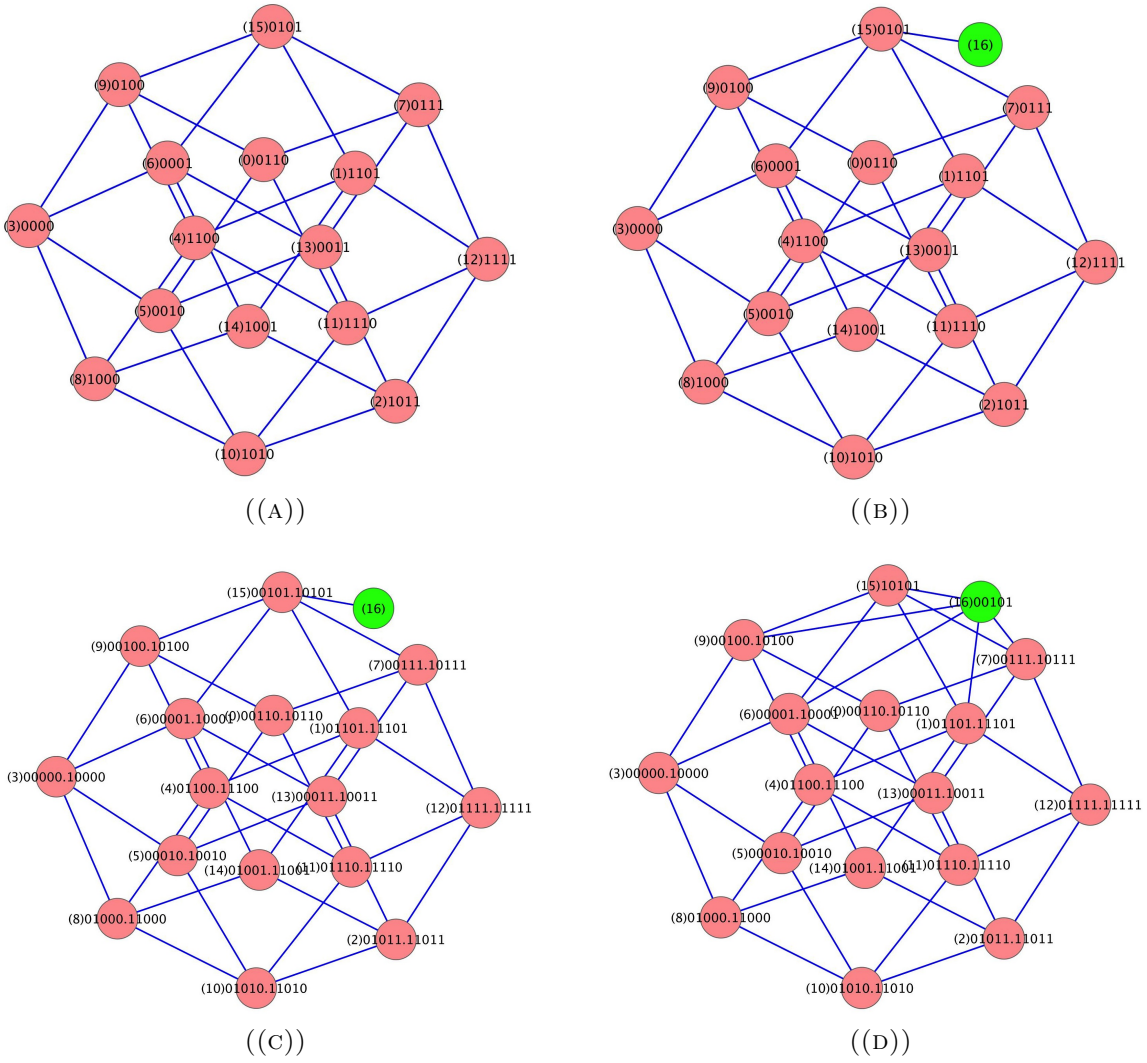

FIGURE 3.8: Example of a Node Entering a Full 4-dimensional HyperD

(15). Additional examples of nodes entering a partial or full HyperD can be found in Appendix A.

Algorithm 5 employs a mechanism of preferential attachment. New entering nodes would first look in their immediate neighborhood for available labels and connect to those nodes that are geographically close when possible. However, as the number of nodes in the network grows, the number of available labels decreases. In some cases a new node is forced to connect to other distant nodes which happen to have some of the few remaining

labels. This may result in significant differences between the overlay and the underlying physical connections. Good communication choices over the overlay may not necessarily translate into comparable savings in the actual physical transmission. To minimize the negative effects of this misalignment, other HyperD variants can be designed. The following two prescriptions exemplify the possible choices for improvement. (1) Limit the enter broadcast range to a fixed number of hops. If a label is not identified within this range then the network would expand to a higher dimension regardless of any labels that may be available at more distant locations in the network. (2) Deploy the initial HyperD at a much higher hypercube dimension, generating enough labels from the start to accommodate a significant number of new nodes.

We implement one of these approaches by limiting the range of enter broadcasts to a single hop. Algorithm 6 shows a modified HyperD enter broadcast. Only the immediate neighbors of the node initiating the broadcast would receive the request message.

Algorithm 6 ModifiedHyperDEnterBroadcast(D, s, c, l_s, l_c)

Input: HyperD Network D, Source Node $s \in V_D$.
Output: Set $A \subseteq V_D$ of nodes with more than one label.

```
1:  A ⇐ ∅
2:  if |LS(c)| > 1 then
3:     A ⇐ A ∪ {c}
4:  else
5:     for each node N_i ∈ neighborhood(c) do
6:        if |LS(N_i)| > 1 then
7:           A ⇐ A ∪ {N_i}
8:        end if
9:     end for
10: end if
11: return A
```

If we limit the number of hops to a small constant value (typically one or two), the resulting HyperD is affected in a number of ways: (1) new nodes will only receive a label from nodes with similar geographic location, thus improving network alignment; (2) the enter procedure complexity is likely to be reduced; (3) HyperD may expand to greater

dimensions even though there are available labels somewhere in the network; (4) the probability that a node accumulates multiple labels increases; (5) node degree distribution is likely in the long run to be less uniform, with some nodes having higher than average number of connections; (6) slight changes may result in routing and broadcast patterns, where more messages may be necessary.

This approach may be a good choice especially in those cases where network changes are approximately uniform over the network. If node enters mostly occur within a local region, multiple network expansions may result. Examples of HyperD networks using both enter broadcast variants can be found in section 3.2.5.2. Charts comparing average path distance in both scenarios are also given.

3.2.4.2 Node Deletions

A deleted node n may announce its departure and inform all neighboring nodes or may fail or leave without notifying its neighbors. In the latter case the deletion is detected during a periodic connection check or during network activity that involves the deleted node. Algorithm 7 describes the steps that are taken when the deletion of a node is detected.

In Algorithm 7 once the departure of n is announced (or detected), the label l with the largest value is selected from the label space of n (Steps 3-7). Node k which owns the label adjacent at the lowest dimension to l is selected to take over the label space of n (Steps 8-10). Node k will then connect to all former neighbors of n (Steps 11-13).

Figure 3.9 shows an example of a node departing a 5-dimensional HyperD. Node (8) with label set 00101.10101 leaves the HyperD network. Node (2) assumes control of the label set of (8) since it owns label 10100 which is adjacent at position 0 to the largest label from (8) 10101. Node (2) upon taking control of the new labels, connects to the

Algorithm 7 HyperDDeparture(D, n)

Input: HyperD Network D, A departing node $n \in V_D$.
Output: A new HyperD D without the departing node n.

```
1: Label l ⇐ −1
2: //Departure of node n is announced or detected
3: for each label l_i ∈ LS(n) do
4:    if l_i > l then
5:       l ⇐ l_i
6:    end if
7: end for
8: Choose l_k|dimension(l, l_k) == 0
9: Choose node k|l_k ∈ LS(k)
10: LS(k) ⇐ LS(k) ∪ LS(n)
11: for each node u ∈ neighborhood(n) do
12:    E_D ⇐ E_D ∪ edge(k, u)
13: end for
14: return D
```

former neighbors of (8), namely (1), (9), (15), (19) and (20). The final result is shown in subfigure 3.9(d).

Remark 3.12. Deletion of a single node only affects local nodes. Each adjacent node given its knowledge of its neighbors' label sets, is able to determine if it needs to assume control of the vacant position following the recent deletion. Most of the incurred cost of this process comes from the redistribution and reconnection of the missing links and the update messages that are sent to announce these changes.

Proposition 3.13. *Let $D = (V_D, E_D)$ be a graph representing a HyperD instance with $2^{n-1} < |V_D| \leq 2^n$.Let$H$=(V_H, E_H) be a n-dimensional hypercube with 2^n vertices serving as a model for D. Let delete(a) be the number of messages sent over the links in D to deal with the deletion of node $a \in D$. Then delete(a) is in $O(log_2(V_D))$.*

Proof. Assume the deleted node a has a label set $LS(a)$ of m labels with m being a small constant value. Then node a maintains mn connections. If node $b \in V_D$ takes control

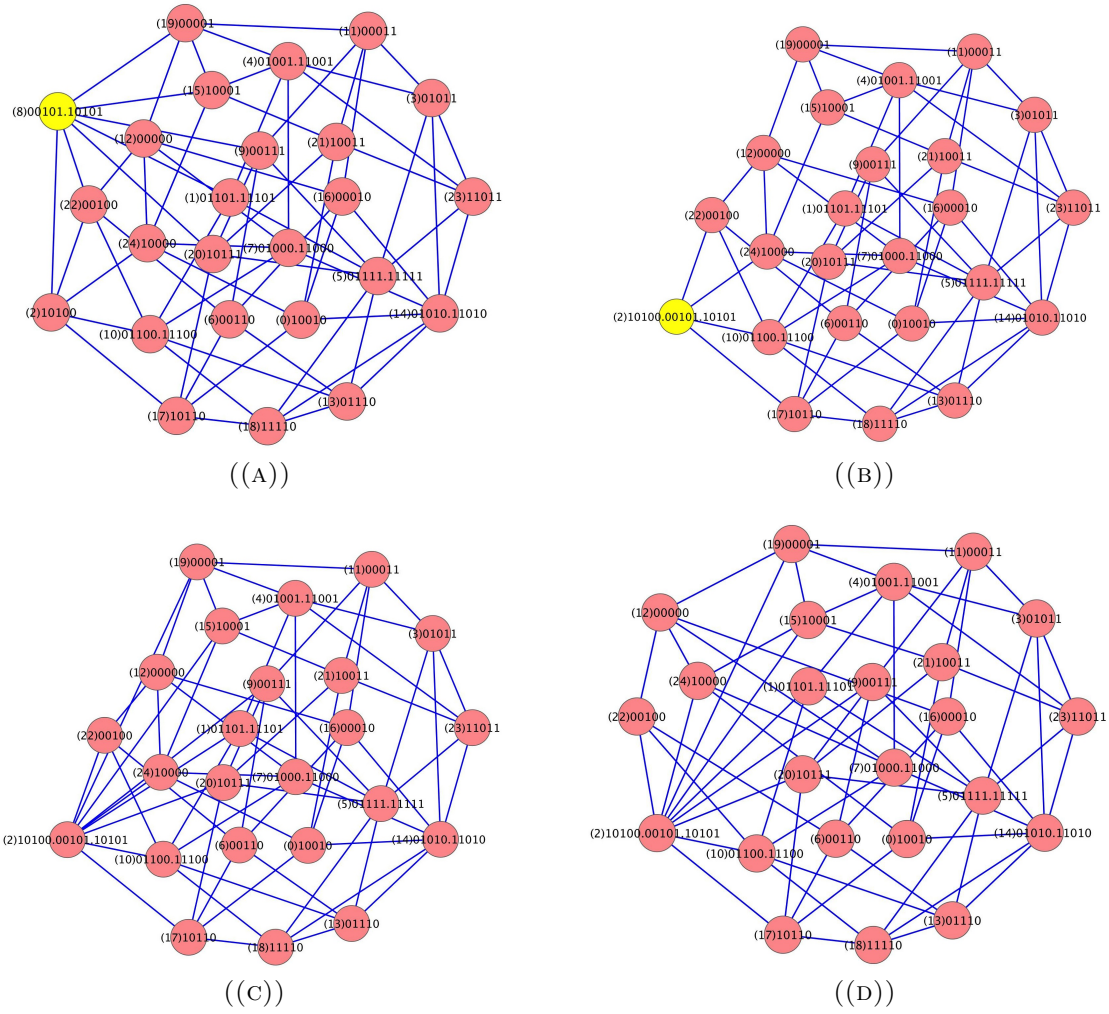

FIGURE 3.9: Example of a Node Departing a Partial 5-dimensional HyperD

of $LS(a)$, then node b needs to connect to at most mn new nodes. Since n is at most $|V_D| + 1$, $delete(a)$ is in $O(log_2(V_D))$. □

3.2.4.3 Other Maintenance Issues

During the existence of a HyperD network, nodes and connections may permanently or temporarily fail. To correctly execute any network activities, each node is responsible to maintain updated information about its neighbors. To accomplish this goal, periodical connection check messages are sent over each link in HyperD. The total cost associated

with this activity depends on the number of links as well as the frequency of these checks. In a hypercube network the number of links per node grows logarithmically in the size of the network. This contrasts sharply with a preferential attachment random network in which the average number of links per node is constant. Therefore it is more expensive to perform this task in a HyperD as the number of nodes increases. To minimize the cost we can reduce the frequency with which such checks are performed. This is particularly feasible if the network is relatively stable. In addition, some of the network changes can be detected during query activity.

3.2.5 Experimental Evaluation

We conduct a number of experiments to illustrate and test the properties of HyperD networks. The goal of these experiments is to: (1) construct and visualize random instances of a HyperD network; (2) simulate and monitor label distribution during the network deployment and label assignment following network changes; (3) show that the distribution is done correctly and is feasible for large networks and to correlate the growth of the label size to the growth of the network size; (4) simulate network activities such as pairwise routing and broadcasting using the assigned labels; (5) assess the efficiency of the routing algorithm by comparing the resulting average path length to the optimal path length of a HyperD instance; (6) perform a global broadcast, monitor that all nodes are reached and estimate the complexity of such broadcasts; (7) test HyperD variations and identify the impact these protocol changes have in the overall performance of the network.

3.2.5.1 Experimental Setup

To create an instance of a HyperD and test all algorithms described in this chapter we coded a JAVA program using the Eclipse platform. Results, such as the set of nodes,

connections between nodes, routing and broadcasting information are printed in files. These files are then visualized using Cytoscape [74, 75] version 2.8.3. Cytoscape is a software tool for visualizing networks. This software specializes in visualizing genomic interaction networks but is well suited for all types of networks. Cytoscape offers a user friendly interface, very neat and appealing visual displays, cross platform compatibility and detailed network statistics.

A number of HyperD networks of various sizes are generated. Each network has an initial size and a final size. We add nodes sequentially until the initial size is met. Each new node entering the network is set to contact a randomly selected node from the existing set of nodes. This node then is used to facilitate the process of finding a new label set and establish new connections. Once the initial size is reached we simulate a number of topology changes which include node enters and deletions. Each one of these two network changes is selected at random based on predefined probabilities. Typically the probability that a node enters is set to be 0.7 to 0.8. So eventually the network will continue to grow until the final size is reached. The final network is then used to test the HyperD algorithms and display the results.

3.2.5.2 Experimental Results

The first example in figure 3.10 shows an HyperD network of 100 nodes. The initial deployment involved 50 nodes. Subsequently, a series of node deletions and enters are randomly simulated. The probability of node enters is set higher than the deletion probability, therefore the network eventually grows to the maximum desired size of 100 nodes. No restrictions are placed on label search. To better understand the distribution of connections we visualize the nodes based on their node degree. Larger sizes and darker color shades indicate larger number of connections for the node. It can be seen that some of the nodes have larger label sets which typically translate into larger node degree. However

differences in node degree are not extreme. The simulation also reveals an underlying uniformity and regularity in the way nodes are connected.

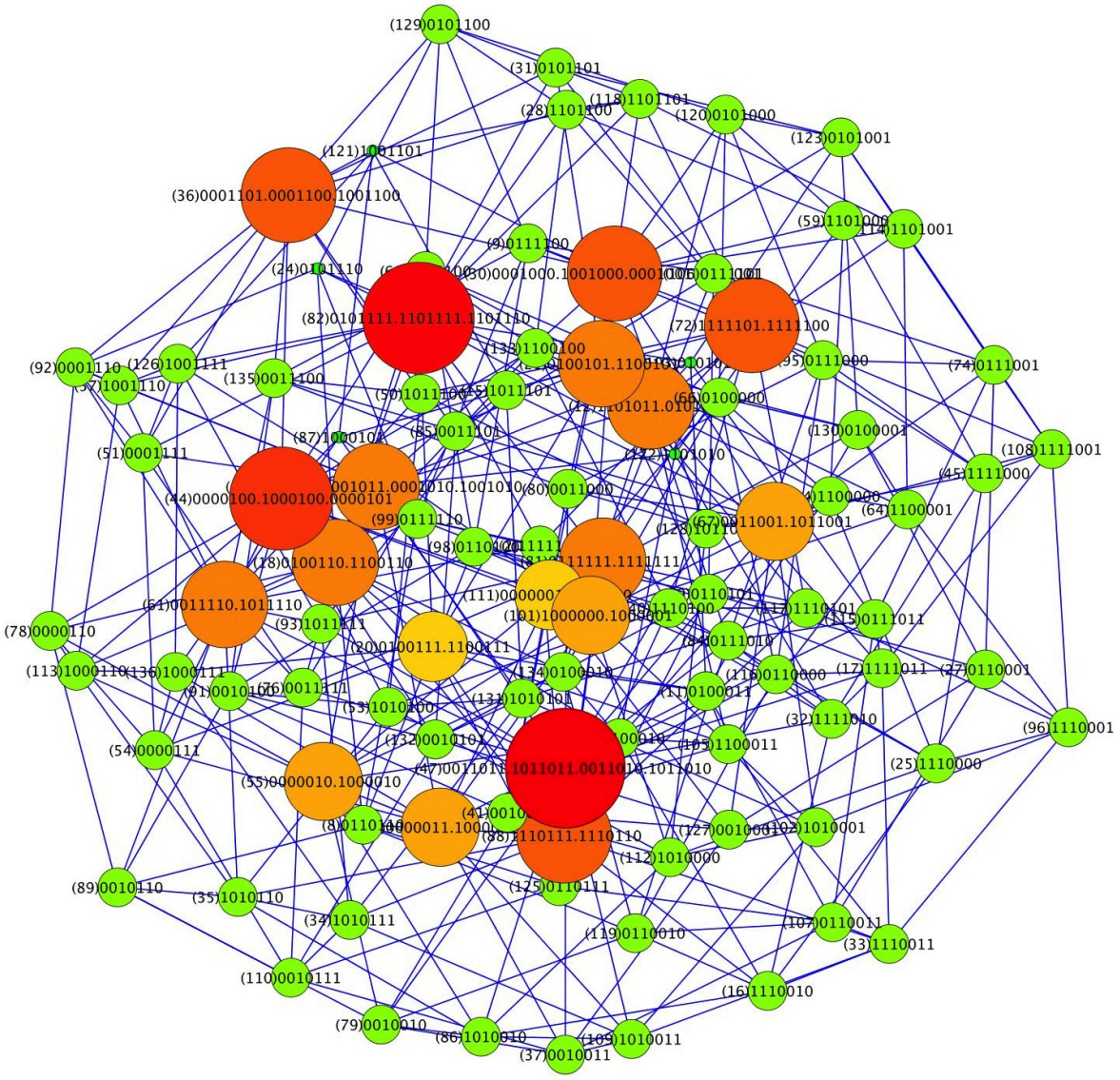

FIGURE 3.10: HyperD Example: 100 Nodes, 50 Start Nodes No Hop Restriction

To better understand node degree distribution, we show a bar chart in figure 3.11. The majority of the nodes maintain 7 connections. This is consistent with the dimension of the underlying hypercube. For those nodes that manage more than one label the degree goes up to 15. For the same network of 100 nodes we simulate finding the shortest path between any pair of nodes. These paths are identified only by using the assigned label

sets and each next hop decision is made locally, involving only neighboring nodes. Results are summarized in figure 3.12.

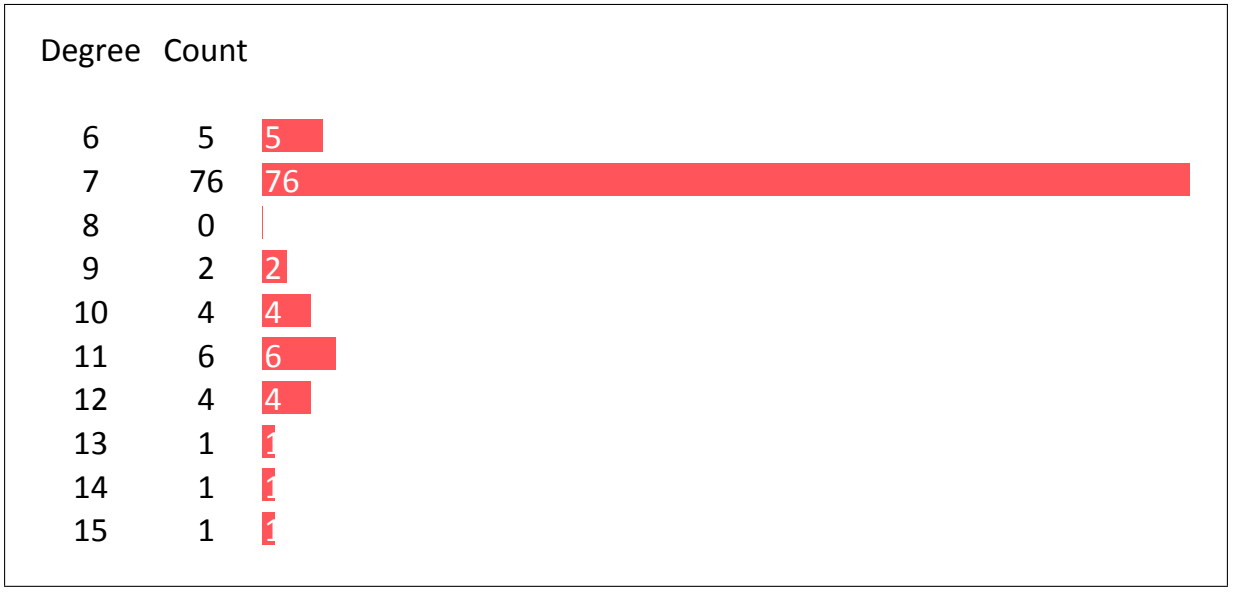

FIGURE 3.11: HyperD: 100 Node Degree Distribution

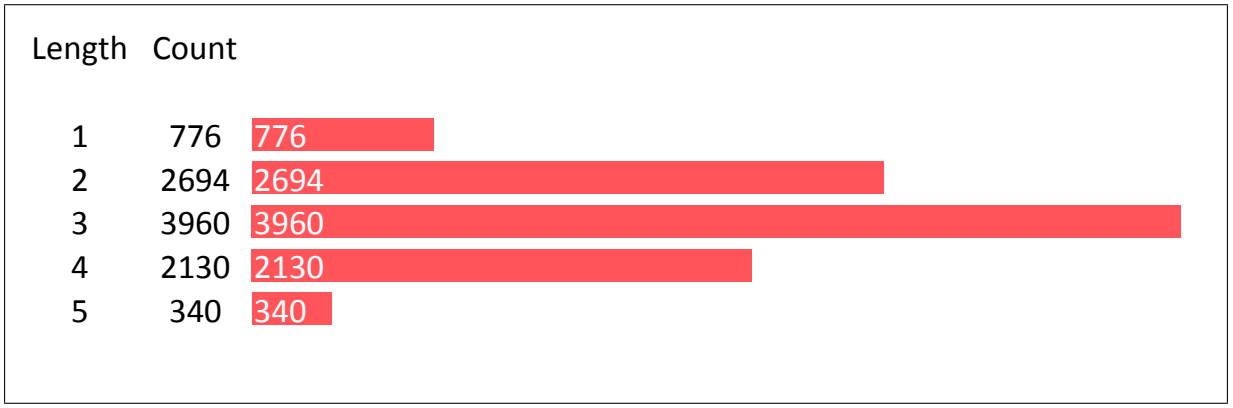

FIGURE 3.12: HyperD: 100 Node Shortest Path Length Distribution

Figures 3.13 and 3.14 compare the average path length of HyperD networks of various sizes. For each HyperD instance we first obtain optimal average path information using the built in features of the Cytoscape software used to visualize the network. We then use the procedure described in algorithm 5 to determine paths between nodes. For reasons summarized in section 3.2.2 there is a very small difference between the HyperD algorithm average and the optimal average. Furthermore, when a restriction on hop count is imposed, we see some small changes in the average path length. However, the difference

is quite small. Details on the data used for these charts can be found in the tables shown in figures B.13 and B.14.

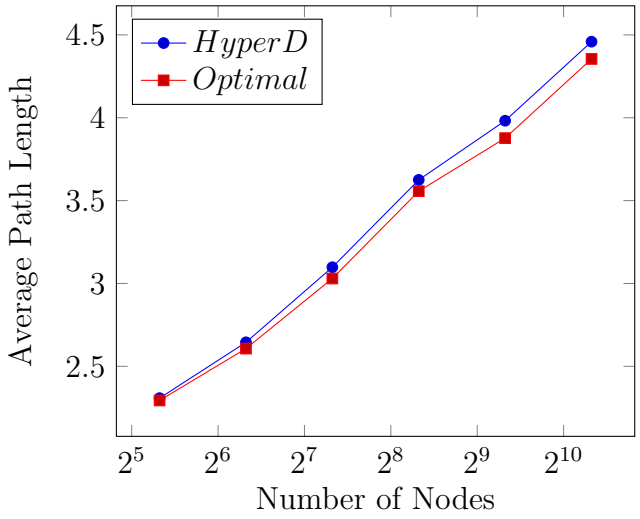

FIGURE 3.13: Average Path Length Comparison No Hop Restriction

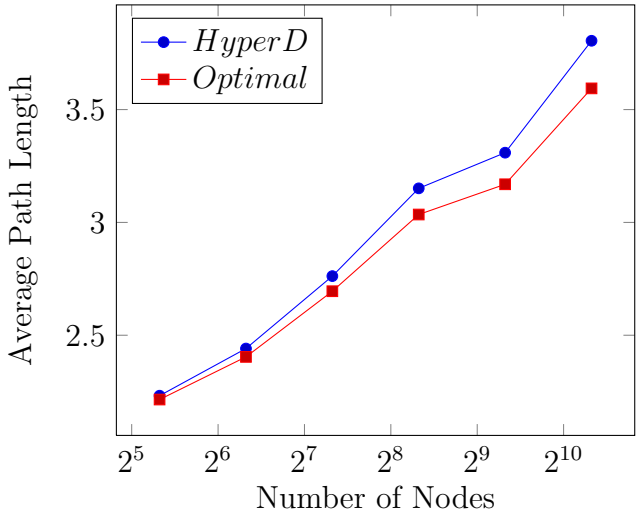

FIGURE 3.14: Average Path Length Comparison One Hop Restriction

Figure 3.15 shows a HyperD network of 200 nodes with no hop restriction on label search. The size and color gradient of each node represent their average path length. Smaller circles correspond to those nodes with better centrality within the network, and are therefore closer on average to the other nodes. We compare the path length distribution obtained in HyperD to the optimal path length distribution. The distribution charts are

shown in figure 3.16. A small difference is noted particularly in the number of longer paths.

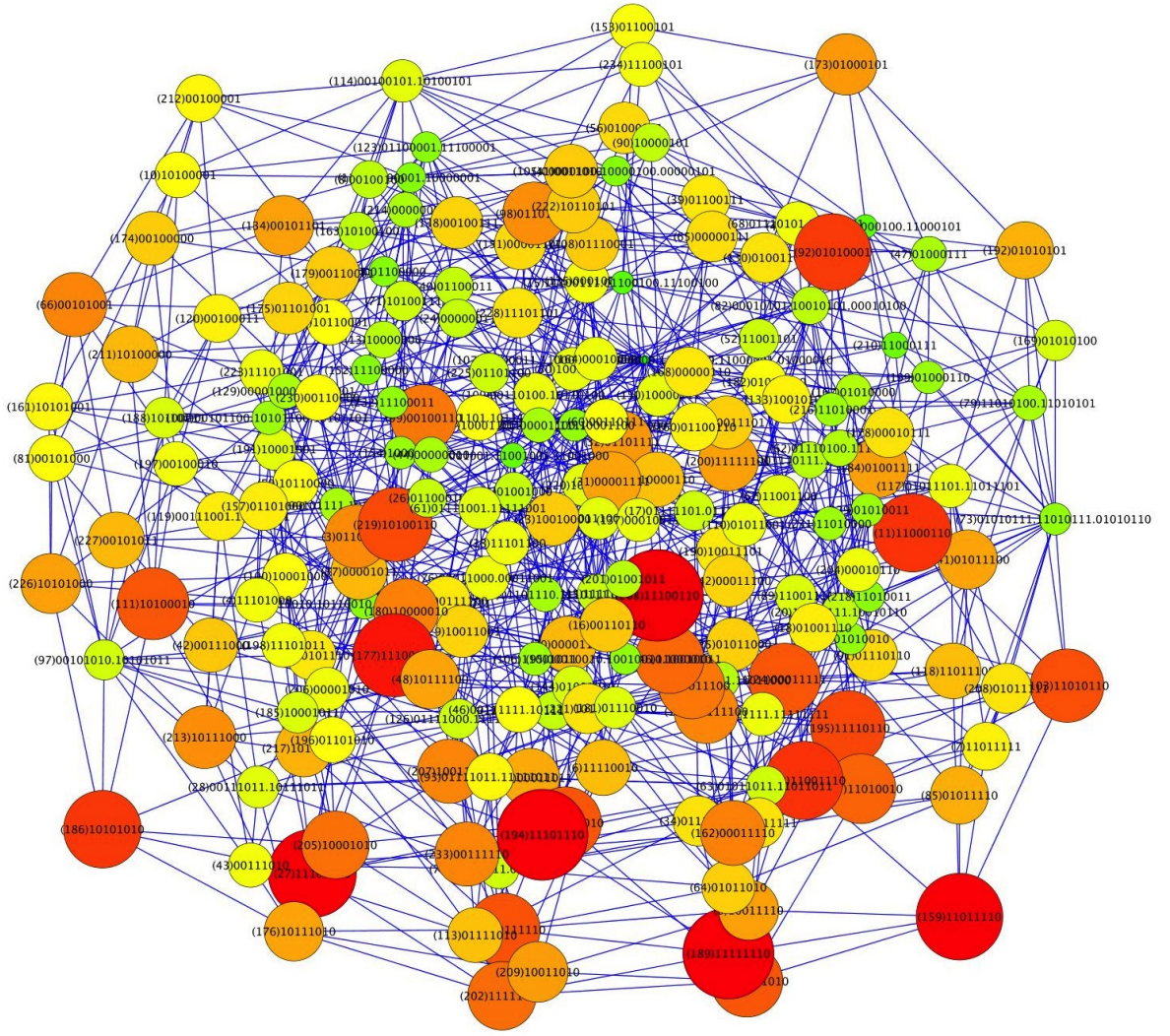

FIGURE 3.15: HyperD Example: 200 Nodes, 150 Start Nodes No Hop Restriction Visualization Based on APL

Figure 3.17 shows a full HyperD of 128 nodes. As expected each node has the same centrality and the same average path length, thus the same visual size. In addition, no difference is noted in path length distributions shown in figure 3.18 since full hypercube algorithms are optimal.

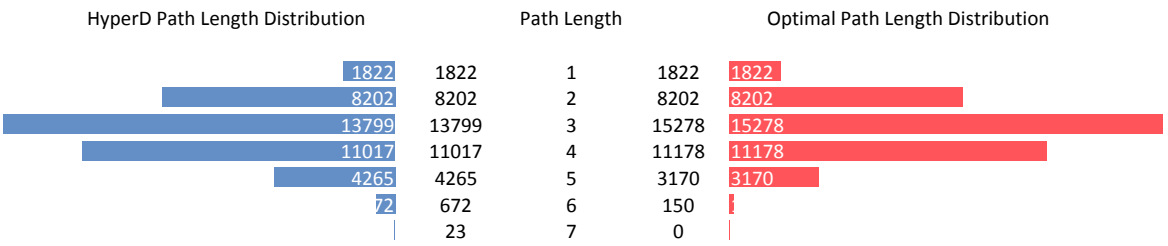

FIGURE 3.16: Path Length Distribution Comparison for HyperD Network in Figure 3.15

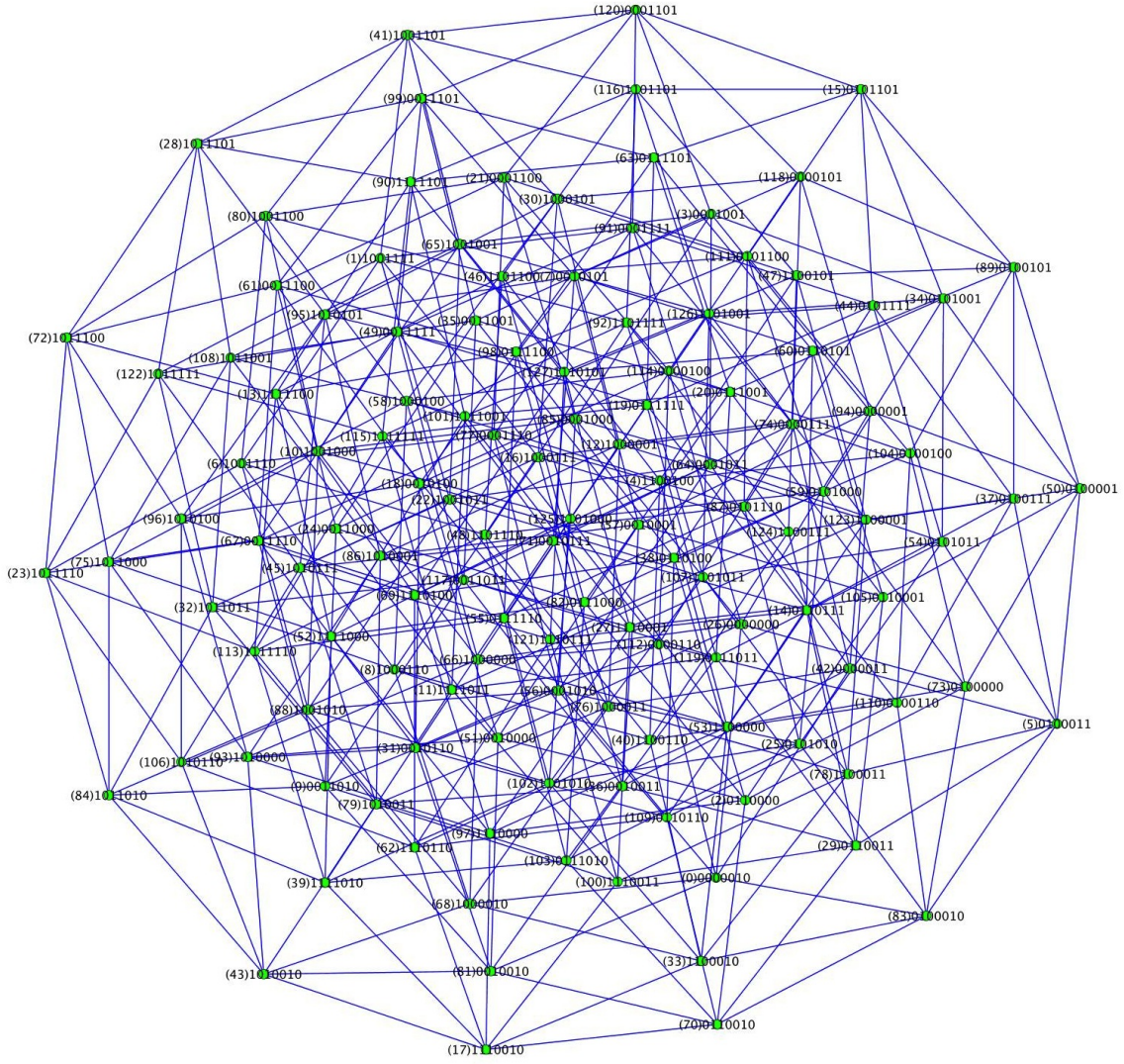

FIGURE 3.17: Full HyperD Example: 128 Nodes, No Hop Restriction, No Topology Changes, Visualization Based on APL

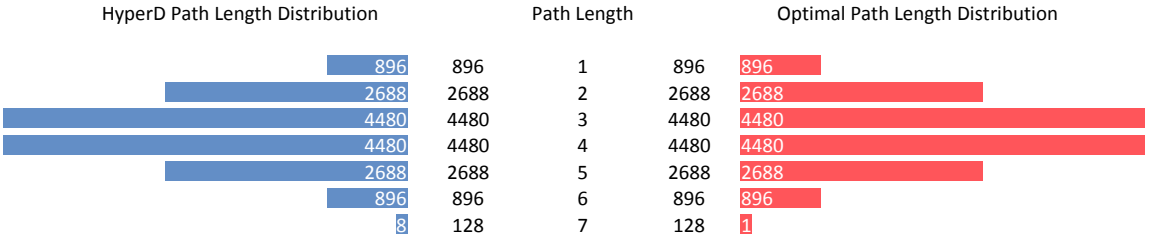

FIGURE 3.18: Path Length Distribution Comparison for the Full HyperD Network in Figure 3.17

Figures 3.19 and 3.20 show a similar example of 200 nodes but with one hop restriction on the enter broadcast. It can be seen that the results do not differ much from those in the example shown in figures 3.15 and 3.16.

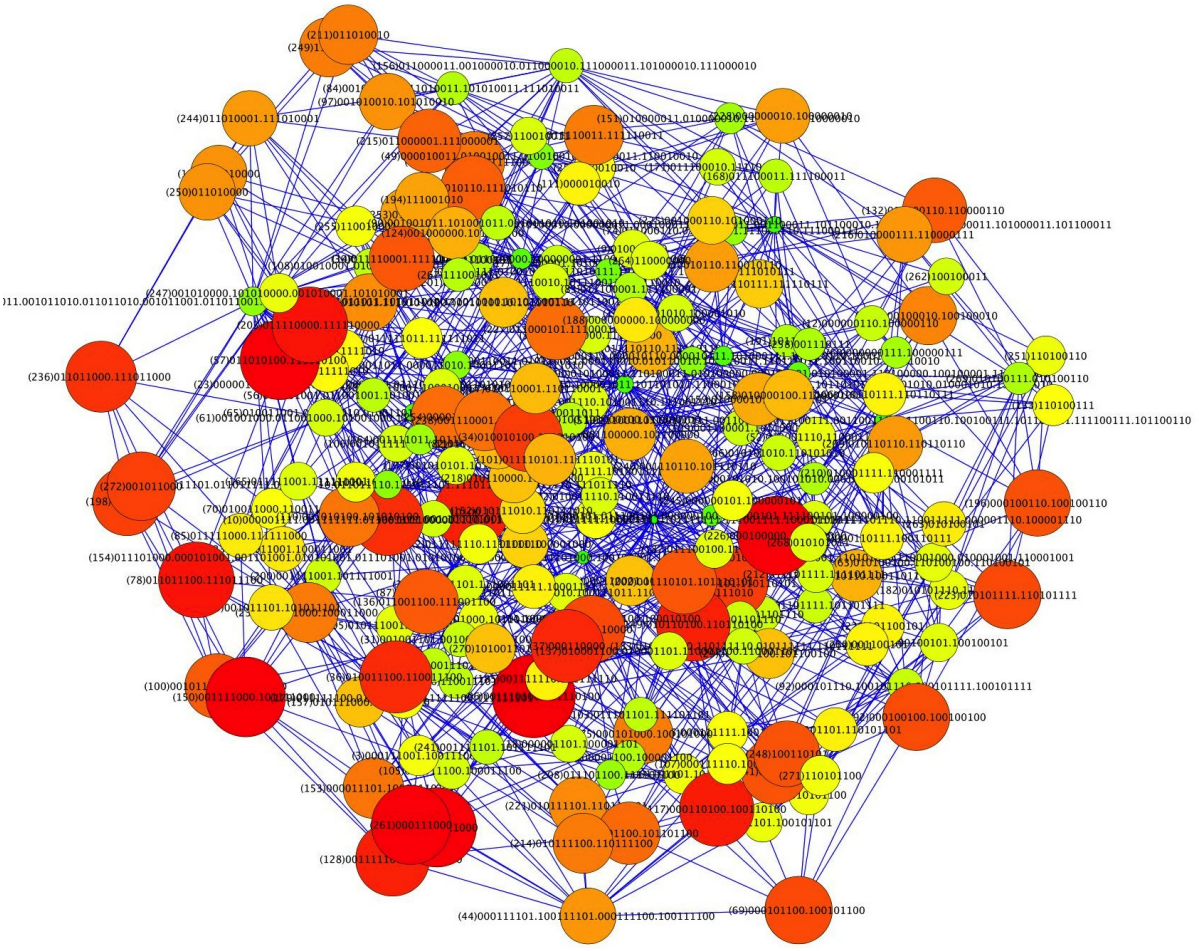

FIGURE 3.19: HyperD Example: 200 Nodes, 120 Start Nodes, One Hop Restriction, Visualization Based on APL

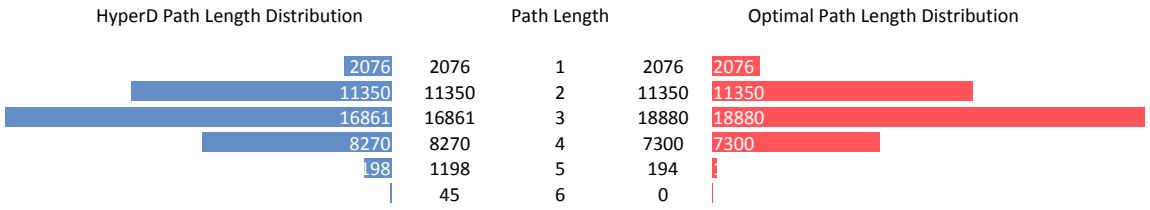

FIGURE 3.20: Path Length Distribution Comparison for HyperD Network in Figure 3.19

Figure 3.22 displays the broadcast tree for the HyperD network shown in figure 3.21. Node size and color depends on the node's out degree. Additional broadcast examples can be found in Appendix B. Figure B.11 is a broadcast tree for a large HyperD of 1500 nodes. Figure B.12 shows the broadcast tree for a HyperD of 800 nodes where visualization of nodes is done using each node's in-degree. It can be seen that a small subset of nodes has received the broadcast message 2 or more times.

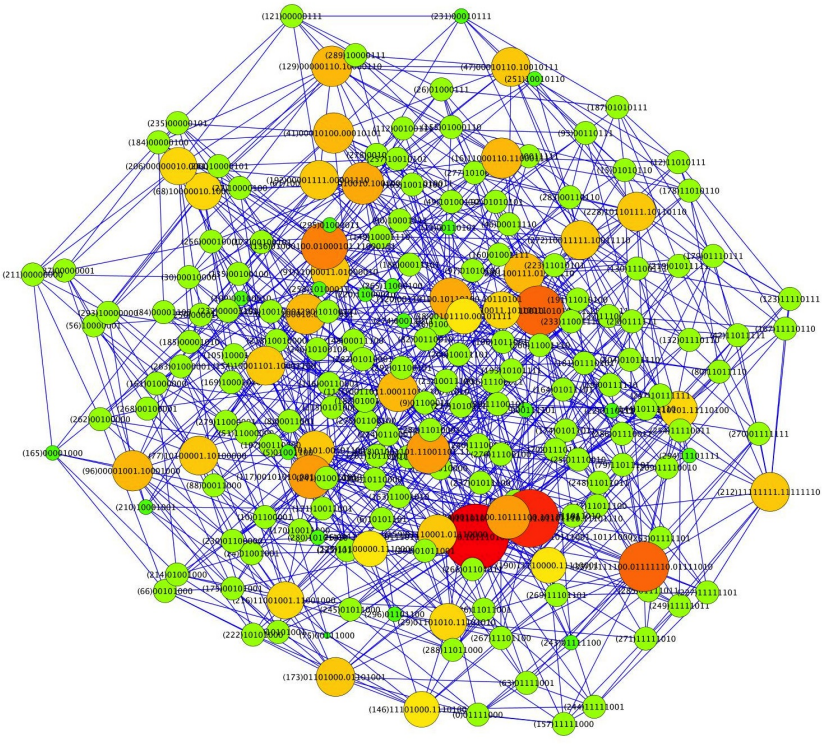

FIGURE 3.21: HyperD Example: 200 Nodes, 100 Start Nodes, One Hop Restriction used to Illustrate Broadcasting

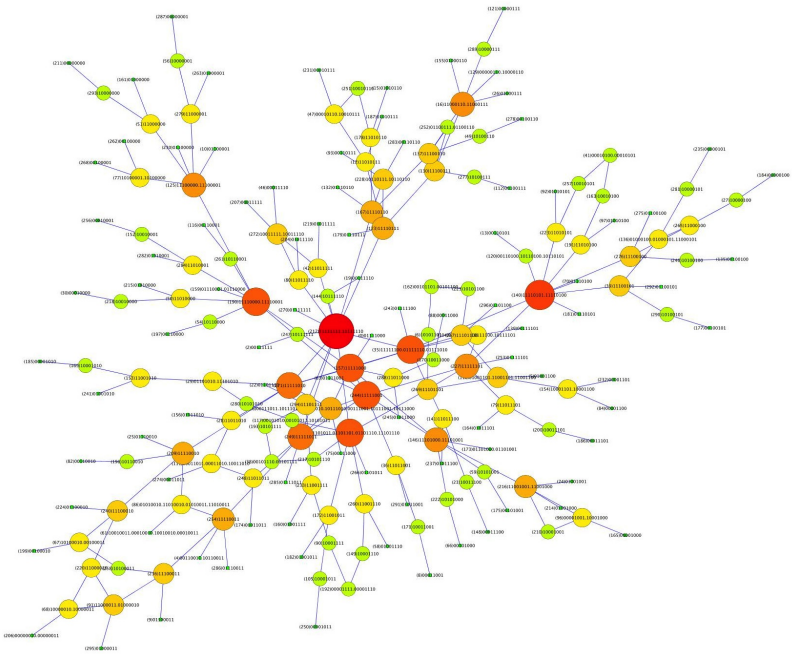

FIGURE 3.22: Broadcast Tree Example for the HyperD Network shown in Figure 3.21, 200 Nodes and 238 Edges

A random HyperD instance of 100 nodes is used to illustrate the distribution of broadcast message counts. A broadcast is issued from each of the 100 nodes and the number of messages for each broadcast is recorded. Results are displayed in the histogram in figure 3.23.

Finally, we test the performance of the broadcast algorithm for HyperD instances of various sizes. We take HyperD networks of sizes ranging from 40 to 1280 nodes. For each size we generate 100 random HyperD networks and for each HyperD we test the broadcast from each node belonging to that network and find the average broadcast message count. A total of more than 250,000 broadcasts from 600 HyperD networks are issued. We summarize the results in a final table shown in figure 3.25 and graph the values in figure 3.24. It can be seen that the number of broadcast messages is nearly optimal for all instances.

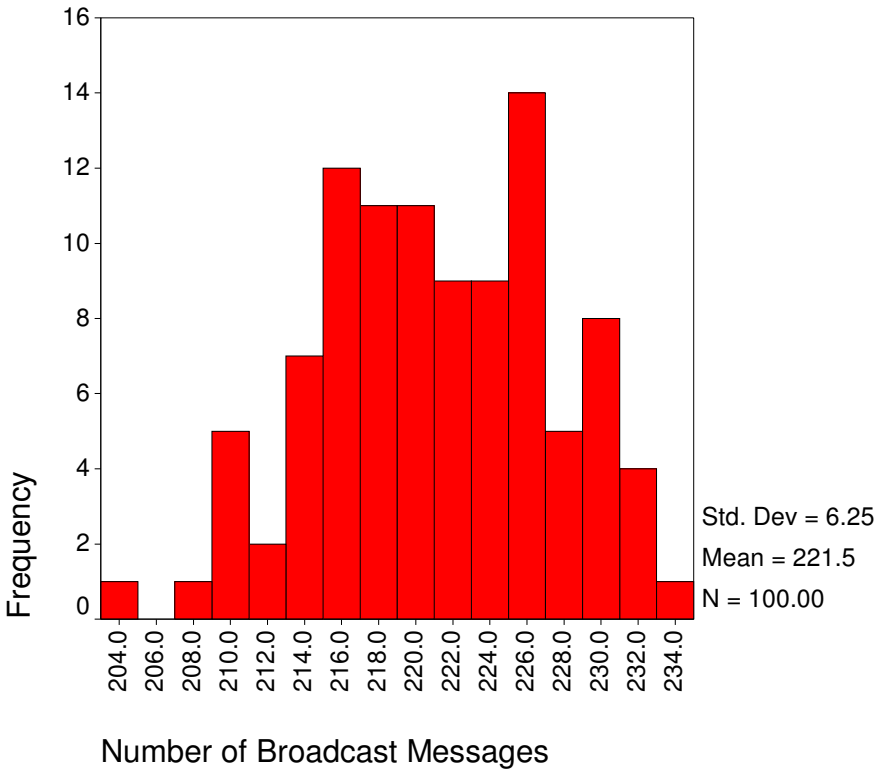

FIGURE 3.23: Histogram Showing the Distribution of Message Counts in a 100 Node HyperD

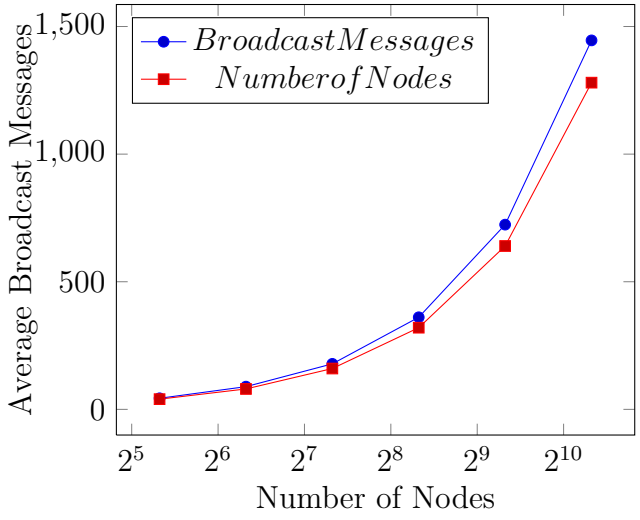

FIGURE 3.24: Average Broadcast Messages for HyperD Networks of Various Sizes

Start Nodes	Total Nodes	Average Broadcast	Percentage Difference
20	40	43.473	8.68%
40	80	89.015	11.27%
80	160	178.331	11.46%
160	320	360.842	12.76%
450	640	723.703	13.08%
900	1280	1445.600	12.94%

FIGURE 3.25: Table Summarizing the Broadcast Count Simulation Results. Values are Used for the Graph in Figure 3.24

3.2.6 Summary

In this chapter we showed that building a hypercube based P2P overlay is not only feasible but proves to be beneficial in terms of reducing the cost of communication. We summarize our observations and list some of the advantages and disadvantages of using HyperD as opposed to unstructured overlay networks.

Advantages: (1) HyperD is relatively simple to deploy and maintain. It takes full advantage of the simplicity, uniformity, regularity and elegance present in hypercube structures. (2) HyperD closely resembles a complete hypercube since all positions are accounted for and managed by some node in the network. (3) HyperD takes full advantage of known optimal hypercube algorithms; (4) Routing is efficient and nearly optimal for most instances. Routes are determined by relying exclusively on the assigned labels. (5) Route discovery is flexible and offers multiple path choices which can prove beneficial in dealing with node failures. Multiple paths can also be used for parallel processing. (6) Broadcasting is nearly optimal, and the number of redundant messages is small relative to the size of the network. Multiple broadcasting trees can be identified to deal with missing nodes and links. (7) All forwarding decisions during communication are made

locally between neighboring nodes. (8) Network changes are typically dealt with at a local level increasing the resilience of the network against high churn. Exception occurs when the network needs to expand to accommodate additional nodes. (9) HyperD is strongly connected and difficult to partition, a property commonly associated with hypercube structures.

Disadvantages: (1) As with any structured overlay we are likely to have a misalignment between the virtual and physical networks. This misalignment can be reduced but cannot be eliminated since the underlying network does not typically resemble a hypercube. (2) The number of connections managed by each node grows with the size of the network. Each connection has a cost associated with its maintenance. (3) There is an additional overhead in distributing and maintaining all hypercube labels. This cost increases particularly in those networks which are highly dynamic and changes in the network are very frequent forcing the nodes to constantly rearrange label distribution. (4) Routing and broadcasting relies on the full cooperation of each peer. In the event that a significant number of peers are permanently or temporarily unavailable, communication may be affected.

Chapter 4

BLWNet: A Binary Labeled Wireless Network

4.1 Introduction

A Mobile Ad-Hoc Network (MANET) is a collection of wireless mobile peers who connect to form a network and communicate and share resources without the help of any centralized authority. We can think of a MANET as being a special type of a P2P network. Wireless networks have attracted considerable attention in recent years. Most wireless technologies such as cellular networks and wireless LANs use a combination of wired and wireless technologies. However, in some specific settings when an infrastructure is not available, the use of MANETs presents a viable alternative. New wireless technologies allow deployment of ad-hoc networks involving a large number of nodes. This is also made more feasible by advancements in mobile technology with more powerful and low cost devices. Examples of wireless ad-hoc networks applications include traffic and weather monitoring, large area security systems and intrusion control. These types of

networks have also been found useful in military operations like military base and terrain monitoring, troop movements detection, and combat communication where military units equipped with wireless devices communicate with each other without central control.

MANETs have been extensively studied. In addition to typical network maintenance issues encountered in wired or centralized networks, MANETs pose a series of new issues. Given the nature of the medium used by these networks it is necessary to consider the limited available bandwidth, device battery capability, interference among nodes sharing the same frequencies, increased vulnerability to attacks, etc.

4.1.1 Motivation

MANETs pose a host of challenges in addition to those faced by traditional P2P networks. The main challenge for MANETs has to do with the characteristics of the medium used by the nodes to communicate with each other. We summarize below the most common problems encountered in MANETs.

Limited Bandwidth Nodes use part of the electromagnetic frequency spectrum which can only offer a limited number of communication channels simultaneously. In addition, those channels currently tend to be underutilized in order to offer dedicated service to some providers. Recent innovations such as cognitive radios, utilize these channels more efficiently; however the physical limitations of the radio frequency spectrum and the increased demand renders this resource an expensive and highly sought after commodity.

Interference Nodes in a MANET would broadcast a message to all neighbors even though only some of them are the target of the message. All nodes within transmission range of each other are sharing the common medium and often a common transmission channel. If nodes are sharing the same communication channel they cannot transmit at the same time without causing interference or collision. Messages may have to be

delayed or repeated which affects the overall quality of service. Interference may also occur between devices operating at different frequency bands.

Dynamicity and Mobility MANETs may be subject to frequent topology changes. Nodes come and go frequently - they can be expected to enter or leave the network at any time. Wireless mobile nodes also change their location continuously causing frequent topology changes. In general, the behavior of wireless nodes can be quite unpredictable.

Limited Resources MANETs are generally characterized by a limited set of resources. Typically a wireless node relies on battery power, has slower processing speed and limited storage capability. These limitations further complicate network stability and reliability and negatively affect the quality of service. Solutions which may work well in typical P2P networks with more stable and powerful nodes may not be feasible for MANETs. We generally seek lightweight solutions which offer reliable service at low cost in terms of resource utilization.

Changes in the Environment Conditions The medium used by wireless nodes may be subject to frequent changes, such as structural obstacles, changes in weather conditions and electrical interference emitted by devices that operate on electrical power. In general, a wireless connection is far less predictable compared with wired connections.

Vulnerability A wireless network is normally more exposed to security threats since the shared medium is accessible to any wireless node including malicious nodes. Any transmission between legitimate nodes can be detected by any node located within its transmission range. A malicious node may cause security issues in a variety of ways such as intercepting data, injecting traffic to overload and disrupt the network, changing the content or the destination of the data transmitted, starving targeted nodes to cause disruption and network fragmentation, etc.

These challenges faced by MANETs have prompted researchers to look for innovative ways to deploy and maintain such networks and find efficient communication strategies. In this chapter our goal is to investigate a new approach which would allow nodes in a MANET to communicate more efficiently. By doing so we aim to reduce the overall traffic, thereby saving valuable resources, reduce interference and increase the quality of service. We propose a new labeling schema to aid message routing in these networks.

4.1.2 Related Work

A great deal of research has been done to improve routing and communication in P2P networks and MANETs in particular. A large number of papers has been published and a variety of solutions has been proposed. In more than one case authors have tried to summarize and classify the different proposals and approaches. Some examples of surveys include [76–79]. A number of overlays designed for wired P2P networks have been extended to MANETs and many other approaches specifically designed for MANETs are also available. Routing protocols for MANETs are of particular interest. An efficient routing protocol serves to reduce the network traffic which in turn improves the quality of service and saves valuable resources. Proposed routing protocols for MANETs differ in a variety of ways. The most common classification of these protocols (adapted from [76]) is shown in Figure 4.1.

Detailed description of these routing categories is widely available in the literature. Here we summarize some of the main characteristics.

Source Initiated Protocols (also known as Reactive Protocols or On-Demand Protocols) When using these protocols, routes are discovered when needed. If a source node does not already know a route to the destination node it initiates a route discovery process, usually by flooding the entire network. The flooding range is in some cases reduced

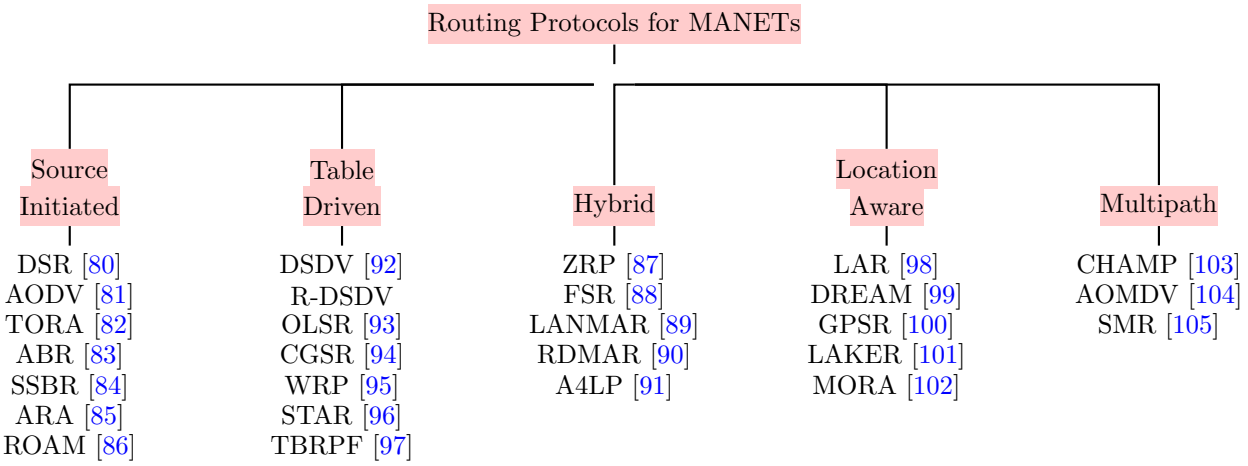

FIGURE 4.1: MANET Classification (adapted from [76])

by using partial data already stored in some of the nodes in the broadcast. The information that is returned from other participating nodes in response to the request is then used to determine the best route. Subsequently the route may be stored for future use for a predefined amount of time or until the route is found to be no longer available. In that case another route discovery process must take place. Source-initiated protocols reduce the overall overhead associated with route discovery since only those routes that are needed are identified. However, the time required to establish a new route increases the end to end communication delay, affecting the quality of service.

Table-Driven Protocols (also known as Proactive Protocols). When a table-driven protocol is implemented each node in the network keeps routing information for all the other nodes in the network. This information is stored in tables maintained at each node. Network and route changes are periodically shared so that each routing table is up to date. These protocols differ in the way updates are sent (complete or partial), in the type of information that is collected about each node and in the flooding mechanisms that are used. These protocols offer better service quality since routes are always available when needed but there is an increased maintenance cost. A large amount of information is exchanged to keep routes current even though these routes may never be utilized.

Hybrid Protocols Source-initiated and table-driven protocols offer possible advantages and disadvantages in specific network scenarios. Hybrid protocols are designed to combine features from both previous approaches, aiming at improving performance. The most typical approach is to proactively identify routes to a set of neighboring nodes, to a cluster or to a network partition and use a reactive approach for nodes that are further away.

Location Aware Protocols (also known as Geographical Protocols) These protocols use the location of the node within the network to determine the transmission routes. This is assuming that all nodes are capable of identifying their location (e.g. using the Global Positioning System (GPS)). In a dynamic environment, node location is periodically updated and changes are normally shared when they occur. Node location is normally used to better target the message to a specific area of the network. A greedy criterion is often adopted to select the next hop to the destination from a set of choices.

Multipath Protocols As the name suggests, these protocols identify multiple routes either proactively or reactively. Using multiple routes increases the reliability of communication. These routes are either used as alternatives in case of failure or to partition the transmitted data for faster delivery.

4.2 BLWNet

In this section we present BLWNet (Binary Labeled Wireless Network), a labeling schema for MANETs. The schema calls for assigning a set of binary labels to each node. These labels are intended to facilitate communication between nodes and simplify route discovery and maintenance. The label assignment does not create an overlay where nodes may be forced to virtually connect to other nodes that may be multiple hops away, causing

a misalignment between the virtual and physical networks. The labels are only used for routing purposes. A node is able to use its labels and those of its neighbors to locally determine which would be the next hop in a point-to-point communication or the next set of nodes in a broadcast activity.

BLWNet does not entirely fit in any of the above mentioned routing protocol categories. It acts as a typical proactive protocol since each node proactively stores information about other nodes in the network. In BLWNet only the label set is stored for each node as opposed to the entire route. This label information is updated periodically or during any normal network activity. When a route is needed messages are forwarded using the labels and any neighboring node information. This is a characteristic of reactive protocols. Furthermore, labels allow for the identification of alternative paths in case of failure and or collisions. This is a typical feature of multi-path protocols. The resulting hybrid protocol offers flexibility for a variety of scenarios.

4.2.1 Model and Preliminaries

4.2.1.1 Network Properties

The ad-hoc network described in this chapter has the following properties:

1. Nodes in the network may be heterogeneous devices which are able to communicate with each other.

2. The network is dispersed and non-trivial, meaning nodes are not all within direct communication range of each other rendering an ad-hoc network unnecessary. A message may need to travel multiple hops to reach a destination.

3. The network is connected. Each node can communicate with all other nodes in the network.

4. The network is dynamic. Nodes may leave, enter, stop working, move within the network, and go into sleep mode periodically.

5. The network is sufficiently dense, where each node has a large number of connection choices (a large number of nodes are within its transmission range). This implies that for the network to remain connected not all wireless links are necessary, allowing us to reduce the transmission volume while preserving connectivity.

6. Each node has the capability to adjust its transmission range by controlling the power used to transmit a message.

4.2.1.2 Model

Let N be a set of n wireless nodes which form a connected ad-hoc network. Nodes from N are located in a bounded two-dimensional region. For simplicity we take the region to be two-dimensional. The protocol can be easily extended to a three-dimensional region. Let $G(G_N, G_E)$ be an undirected graph modeling the network where G_N with $|G_N| = n$ represents the set of nodes from N, and G_E with $|G_E| = e$ represents the set of all possible edges (bidirectional wireless links) between the nodes in N. Note that a bidirectional wireless link exists between two nodes if they are able to transmit to each other using their maximum allowed transmission power. Let the label set LS_d be the set of all possible binary numbers with d digits. We can think of each of the labels $l_i \in LS_d$ as one of the vertices in the d-dimensional hypercube Q_d. The goal is to assign to each node $v_i \in G_N$ a set of binary labels $LS(v_i) \subseteq LS_d$. We can think of node v_i as being assigned a portion of Q_d. We then use the assigned labels to reduce the overall communication cost, reduce

energy consumption and decrease interference by using fewer links during routing and broadcasting and by dynamically adjusting the transmission range of each node.

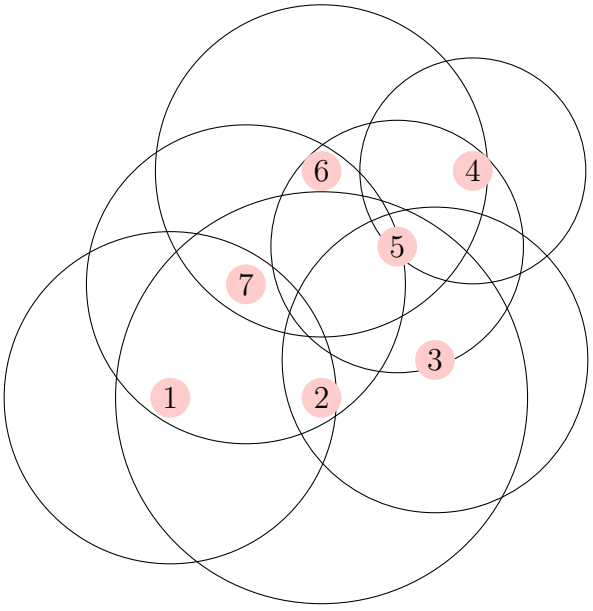

FIGURE 4.2: An Example of a Wireless Network of 7 Nodes

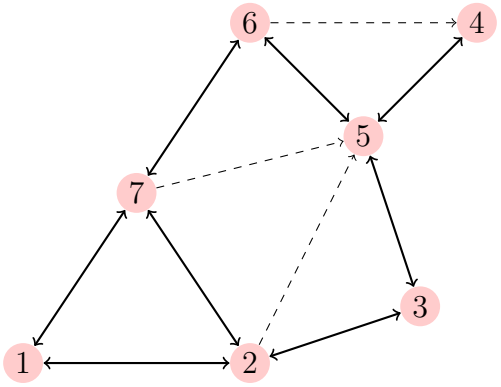

FIGURE 4.3: An Example of a Directed Graph Modeling a Wireless Network of 7 Nodes

Figure 4.2 shows an example of a wireless network of 7 nodes. The transmission range of each node is represented by a circle centered at the node. For simplicity we assume Line of Sight (LoS) propagation model [106] and take these shapes to be circles. In practice, shapes could be different based on the specific transmission capability of each node and the conditions of the surrounding environment. The analysis can be easily extended to each scenario. Figure 4.3 shows a directed graph indicating all possible links between

any pair of nodes in the network. The bidirectional edges are drawn between two nodes which can send and receive messages between each other. The directed dashed lines indicate that one of the nodes can transmit a message but is not able to receive due to range limitations. If we consider the network in figure 4.2 to be N, then G is the graph from figure 4.3 once the dashed lines are removed. A larger example can be found in appendix C.

Each node $v_i \in G$ has a set of neighboring nodes denoted by $neighborhood(v_i)$. The degree of a node v_i, $degree(v_i)$ is equivalent to the size of $neighborhood(v_i)$. Therefore $degree(v_i) = |neighborhood(v_i)|$. If during the label assignment process node v_i donates a label set to one of its neighbors v_j we say that v_i is $parent(v_j)$ and v_j is $child(v_i)$.

4.2.1.3 Label Format

A label is a string containing the symbols 0 and 1. The label set $LS(v_i)$ for a node $v_i \in G$ is the set of all such strings assigned to v_i. Each label string has a finite number of digits and represents all possible binary numbers ending with those digits (e.g a node is assigned the string 1001, the node owns all possible binary strings ending with 1001). We can think of a node as owning a portion of all possible binary strings.

Definition 4.1. The dimension of a node v_i $dimension(v_i)$ is equal to the size of the smallest label owned by node v_i (e.g. if a node v_i is assigned the binary string 1001101 then $dimension(v_i) = 6$).

We use a small example in Figure 4.4 to illustrate the partition of the available labels. Assume we use the 3-dimensional hypercube Q_3 as the maximum capacity of the network. Q_3 has $2^3 = 8$ available positions (labels). These 8 labels are shared among the nodes in the network. When the first node is assigned a label set the node will own all 8 available labels. As other nodes enter the network the labels are shared accordingly.

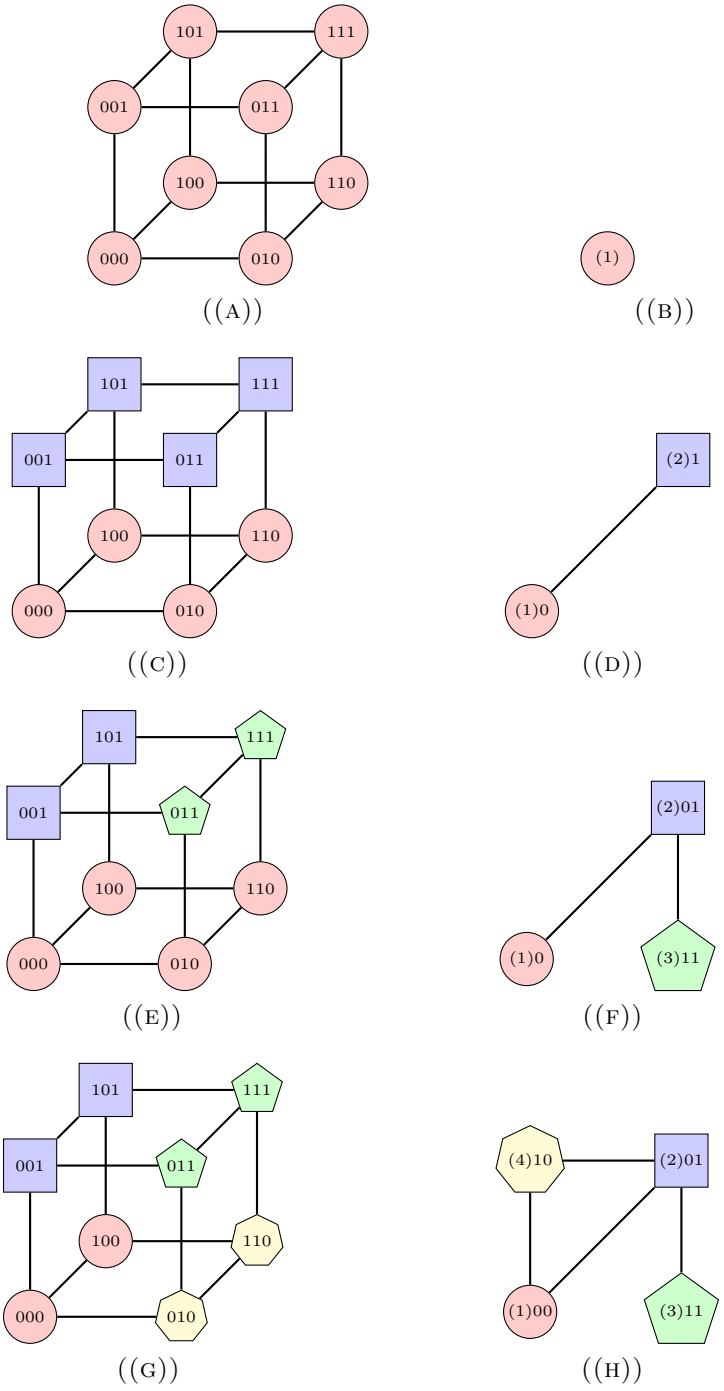

FIGURE 4.4: Label Partition Example

Node (1) is the first to enter and will be assigned all 8 available labels (Figures 4.4(a) and 4.4(b)). When node (2) enters, it receives half of the labels from node (1). Now node (1) owns all those labels ending in 0 and node (2) all those labels ending in 1 (Figures 4.4(c) and 4.4(d)). Node (3) will receive labels from node (2). Node (2) now has all labels ending in 01 and node (3) those labels ending in 11 (Figures 4.4(e) and 4.4(f)). Finally, node (4) will receive the label set 10 from node (1). Node (4) now has all labels ending in 10 and node (1) those labels ending in 00 (Figures 4.4(g) and 4.4(h)). Even though we think of these labels as belonging to a 3-dimensional hypercube, they are not used to determine network connections. The only connections maintained by BLWNet are between any two physical neighbors. Thus we do not form an overlay network. As a consequence we may have two nodes owning adjacent labels but not being connected. Conversely two neighboring nodes may not own adjacent binary labels.

In practice, a network could be significantly larger, with thousands or more participating peers. In such cases the chosen maximum hypercube dimension has to be sufficiently large to accommodate these nodes.

4.2.2 BLWNet Deployment

Given an initial set of wireless nodes belonging to a wireless network N, let the nodes follow the steps shown in Algorithm 9 for deploying a BLWNet instance modeled by G.

Definition 4.2 (Label Set Partitioning)**.** Let m be a node from a MANET M which already has a label set assigned. Then $PartitionLabels(m)$ describes the process when node m donates a portion of its label set to those neighbors which do not have any labels and have not yet identified another donor node.

$PartitionLabels(m)$ (Algorithm 8) is a recursive method. Each node will recursively continue to partition its labels until all their neighbors have received a portion of the

label space. This procedure is invoked when the network is first deployed or when new nodes enter the network. When BLWNet is first deployed the method calls will recursively flood the network until all nodes are assigned their label set.

Algorithm 8 $PartitionLabels(m)$

Input: Graph $M(V_M, E_M)$ representing a connected MANET, node $m \in M$.

$NeighborsWithoutLabels \Leftarrow \emptyset$
for each node $m_i \in neighborhood(m)$ **do**
  if $LS(m_i) == \emptyset$ **then**
    $NeighborsWithoutLabels \Leftarrow NeighborsWithoutLabels \cup m_i$
  end if
end for
Split $LS(m)$ until number of splits is greater than $|NeighborsWithoutLabels|$
for each node $m_j \in NeighborsWithoutLabels$ **do**
  $LS(m_j) \Leftarrow$ one of the partitions of $LS(m)$
  $PartitionLabels(m_j)$
end for

Example 4.1. *Let $m_0 \in M$ be a node with label set $LS(m) = 101$. Assume m_0 agrees to partition its label set with two of its neighbors (m_1 and m_2) which need to obtain a new label set. We can split in two the label set by simply adding (padding left) the current label set with a 0 and a 1. If necessary we can split the set in four by adding two digits, split in eight by adding three digits and so on. For this example we need at least three parts. Node m_0 partitions its label set in four subsets (00101, 01101, 10101, 11101). Node m_0 then donates subsets 10101 and 11101 to nodes m_1 and m_2 respectively. Now $LS(m_0) = 00101, 01101$, $LS(m_1) = 10101$ and $LS(m_2) = 11101$.*

Algorithm 9 describes the process of deploying BLWNet. Before this algorithm is implemented each node v_i sends beacon messages including its ID using its maximum transmission power. Based on the responses, v_i identifies its neighbor set $neighborhood(v_i)$. An edge is added to graph G from v_i to any of its new neighbors. Once all nodes connect to their neighbors, a node v_0 is elected to be the first node to obtain a label. The election process can be done following the Highest Degree Algorithm [107]. The node with the

Algorithm 9 DeployBLWNet(G, d)

Input: Graph $M(V_G, E_G)$ representing a connected MANET, hypercube dimension d.
Output: Labeled M.

1: $T \Leftarrow \emptyset$
2: **for** each node $m \in G$ **do**
3: Calculate node degree
4: **end for**
5: Select the node(s) with highest degree
6: **if** more than one choice with highest degree **then**
7: Select the node with lowest ID
8: **end if**
9: // Let node n_0 be the selected node
10: $LS(n_0) = LS(Q_d)$
11: $PartitionLabels(n_0)$

largest number of neighbors is selected. Ties are broken using the lowest node ID (Steps 2-9). Node v_0 is assigned all d-digit binary numbers corresponding to all positions of Q_d (Step 10). Once the first node to join BLWNet is selected the $PartitionLabels(n_0)$ method is invoked which recursively partitions the label set among the nodes in the network.

Figure 4.5 shows the result of applying Algorithm 9 to a network of 7 nodes. There are a few nodes with degree 3. Node (2) is the first selected node since it has the lowest ID value (Subfigure 4.5(b)). Subfigure 4.5(e) shows the final label partition.

4.2.3 BLWNet Maintenance

In this section we describe two basic maintenance protocols for when a node enters BLWNet or leaves the network.

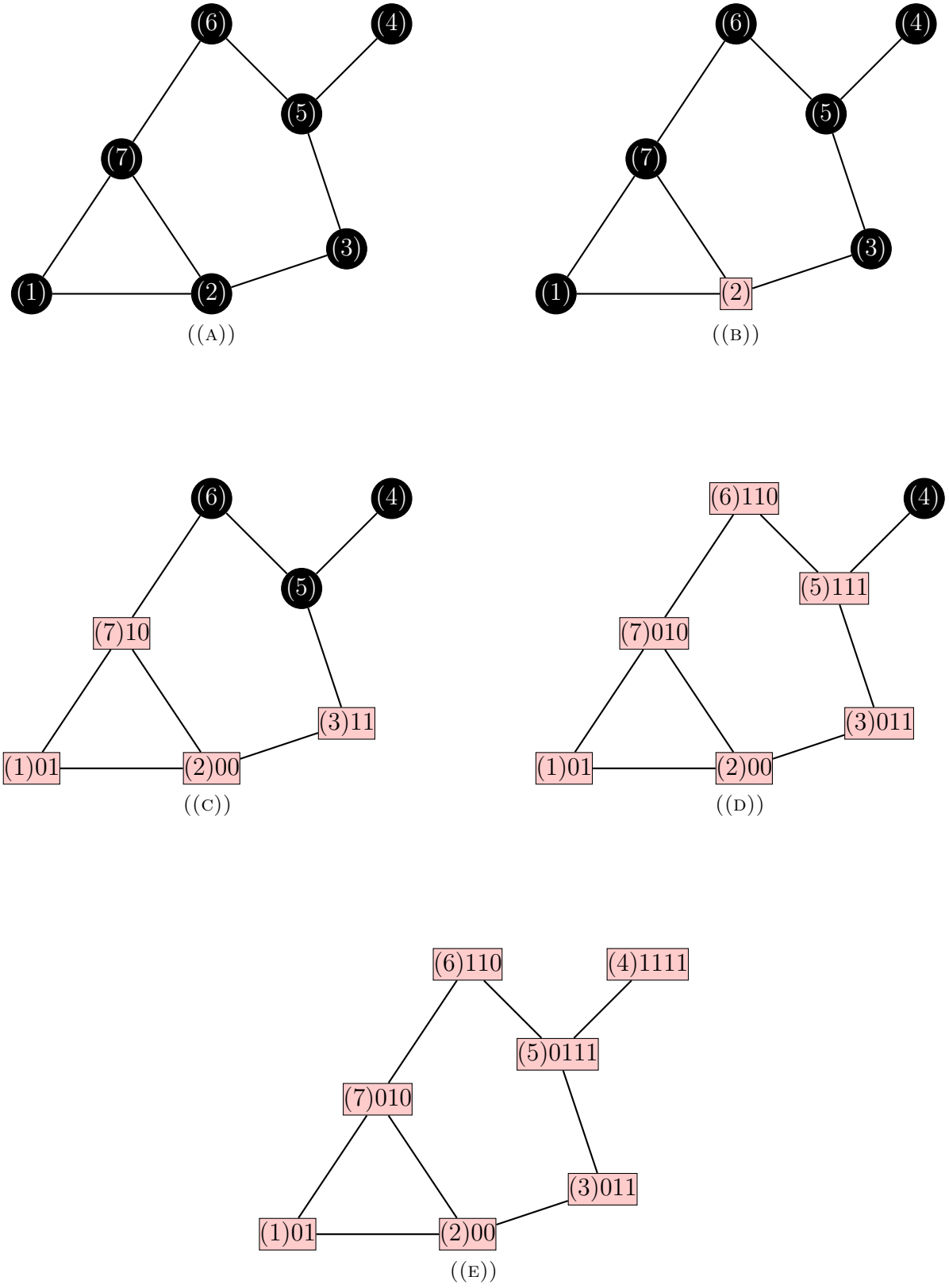

FIGURE 4.5: Deployment of a BLWNet

4.2.3.1 Node Enters

Algorithm 10 describes the steps taken to add a new node to a BLWNet network. When a new node m requests to enter a BLWNet network, it first needs to identify all those nodes from G that are within its communication range. Node m connects to each of these neighboring nodes (Steps 2-4). Once all connections have been established the new node m needs to obtain a new label set. Node m chooses the node with lowest dimension m_c from its neighbors. As previously mentioned the dimension of a node v_i, $dimension(v_i)$ is equal to the size of the smallest label in $LS(v_i)$ (Step 5). Ties are broken by choosing the node with the lowest ID. Node m_c shares its label set with m (Step 6).

Algorithm 10 EnterBLWNet(G, m)

Input: Graph $G(V_G, E_G)$ representing a connected MANET, node m.
Output: G including node m.
1: $neighborhood(m) \Leftarrow \emptyset$
2: **for** each node $m_i \in G$ within the range of m **do**
3: $\quad neighborhood(m) \Leftarrow neighborhood(m) \cup m_i$
4: **end for**
5: Choose neighboring node m_c with smallest dimension
6: $PartitionLabels(m_c)$
7: **return** G

Figure 4.6 shows an example of a node entering a BLWNet of 7 nodes. Node (8) first connects to its new neighbors (3), (4) and (5) from the existing network (Subfigure 4.6(a)). Node (3) is selected to share the label set with (8) since $dimension((3)) = 2$ compared with $dimension((4))$ and $dimension((5))$ which are 3. Node (3) finally shares its label set with (8) (Subfigure 4.6(b)).

4.2.3.2 Node Deletions

When a node decides to leave the network, or if a node stops functioning, a procedure is followed to maintain the correct label distribution and preserve the necessary hierarchical

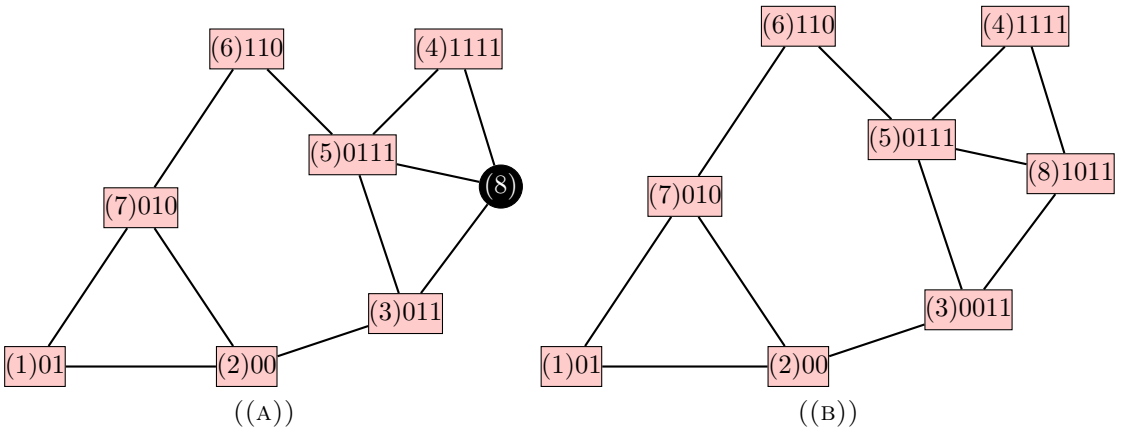

FIGURE 4.6: Example of a Node Entering BLWNet

relations between the nodes.

Before we present an algorithm for node deletions we need to discuss some necessary notations and definitions.

Definition 4.3 (Ancestry Tree)**.** Let G be the graph representing the network and let n_0 be the first node to obtain a label. Then $ancestryTree(G)$ is the subgraph of G containing all nodes from G and all parent-child links. The $ancestryTree(G)$ is therefore a tree rooted at n_0 with branches that follow the parent-child relationship. Figure 4.7 shows an example of an ancestry tree for a BLWNet of 7 nodes.

In the event of a node deletion there are two issues that need to be addressed: (1) Which node will get the label set from the deleted node? It is desirable that labels are recycled following the deletion of a node. However, the network can function even in the event some of the labels cannot be recovered. (2) What does a child node do upon the deletion of its parent node? It is necessary for the child node to identify a new parent node in order to preserve the integrity of the ancestry tree.

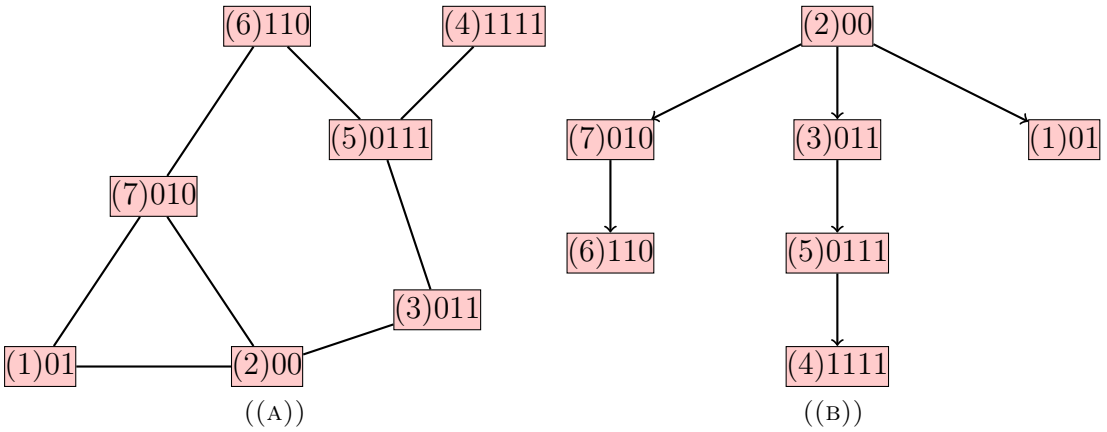

FIGURE 4.7: Example of the Ancestry Tree

Finding a new parent is the most complex decision a node has to make in the event of a node deletion. Algorithm 11 shows the steps taken when the deletion of a parent node is detected by one of its children nodes. The following definition of 'label match' is needed here.

Definition 4.4 (Label Match)**.** Let $match(s_1, s_2)$ be the size of the perfect rightmost digits match of two binary strings s_1 and s_2. Let u and v be two nodes belonging to a BLWNet. Then $labelMatch(u, v)$ is the maximum $match(l_u, l_v)$ for any label $l_u \in u$ and any label $l_v \in v$. For example, $labelMatch((2)1011.1001), (3)0001) = 3$ since $match(1001, 0001) = 3$.

In Algorithm 11, two variables ($temporaryParent$ and $currentParentMatch$) are initialized to record the best choice for parent selection as the algorithm goes through various iterations (Steps 1-2). Node d, looking for a new parent connection, first tries to identify a neighboring node that is not the current parent or one of its current children (Steps 3-8). If no such alternative is available, node d would choose one of the children to swap positions and become the new parent node. This is only done after the selected child

node is itself able to identify a new parent (Steps 9-18). If the alternative parent node is indeed identified, the new parent-child relation is established (Steps 19-22).

Algorithm 11 findNewParent(B, d)

Input: BLWNet network B, A node $d \in V_B$ that needs to identify a new parent.
Output: TRUE if d finds a new parent, FALSE otherwise.

```
1: Node temporaryParent with temporary id -1;
2: int currentParentMatch ← 0;
3: for each node u_i ∈ neighborhood(d) do
4:    if u_i is not parent(d) && u_i is not child(d) && labelMatch(u_i, parent(d)) >
      currentParentMatch then
5:       temporaryParent ← u_i;
6:       currentLabelMatch ← labelMatch(u_i, parent(d));
7:    end if
8: end for
9: if temporaryParent.getId() == −1 then
10:   for each node c_i.child(d) do
11:      if findNewParent(c_i) then
12:         d.setParent(c_i);
13:         d.children.remove(c_i);
14:         c_i.addChild(d);
15:         return true;
16:      end if
17:   end for
18:   return false;
19: else
20:   d.setParent(temporaryParent);
21:   temporaryParent.setChild(d);
22:   return true;
23: end if
```

Algorithm 11 is an essential prerequisite for Algorithm 12. Algorithm 12 describes the steps taken when a node is deleted. Here is a brief summary. The parent of the deleted node would assume control of the label set (Step 1) and remove the deleted node from its children list (Step 2). Each of the children of the deleted node need to find a new parent to avoid disconnecting the ancestry tree (Steps 3-5). The deleted node and all its adjacent links are removed from the network (Steps 6-7).

Algorithm 12 deleteBLWNetNode(B, d)

Input: BLWNet network $B(V_B, E_B)$, A node $d \in V_B$ leaving B.
Output: B after deletion of d.

```
1: LS(parent(d)) ← LS(parent(d)) ∪ LS(d));
2: parent(d).children.remove(d);
3: for each node c_i.child(d) do
4:     findNewParent(c_i);
5: end for
6: V_B ← V_B − d;
7: E_B ← E_B − d.getLinks();
8: return B;
```

Remark 4.5. If two or more adjacent nodes are deleted simultaneously it may be difficult to recover and 'recycle' all the label sets. In particular, if a parent-child pair is lost, then the child's labels which were supposed to be recovered by the parent node will be lost. The absence of a set of labels from the total network label space will not compromise communication in BLWNet since labels are not used to determine network connections but are only used as tools to identify communication paths. For details on how routing works in BLWNet refer to section 4.2.4.1.

Remark 4.6. In the event the root node is deleted, one of its children would take control of its labels and the position as the root of the ancestry tree. The choice of a successor can be made immediately or when needed based on a predefined criterion.

Remark 4.7. As an alternative to preventing the loss of a label set, nodes can maintain information not only about their immediate neighbors but also those neighbors which are two or perhaps more hops away. This would allow nodes to successfully recover labels from multiple node deletions such as in the case of a parent-child pair. Label recovery may increase the stability of the label structure and make routing decisions more accurate. However, this approach would require additional overhead and more frequent updates to be shared among nodes, and may incur greater cost than the loss of some labels. Continuous changes in the topology and the redistribution of the labels may cause some labels to be lost over the lifetime of a BLWNet. In addition, imbalances

in label distribution may also occur (e.g node accumulating large label sets and some other nodes perhaps only owning small label sets.) In order to address these issues a periodical redeployment of the network may be beneficial. Among other specifications in the communication protocol, nodes are required to participate in the redeployment of the network and a fresh distribution of labels. The frequency of such redeployments may be determined based on the specific nature of a network. In a highly dynamic network these may need to occur more often.

Partial Relabeling When a node is deleted, a child looking for a new parent may be forced to convert one of its own children nodes into its parent node (see Algorithm 11). Reversing a connection from parent-child to child-parent may result in less accurate routing decisions. To maintain a better label structure, it may be beneficial to relabel a subset of nodes. The steps needed for partial relabeling following the deletion of a node are given in Algorithm 13. If a node sets as its parent one of its former children nodes, it gets a label from that node. Once the label is assigned the redistribution starts from the newly relabeled node (Step 3). The node would then assign a new label to each one of its current children, if any. In turn all children would recursively repeat the label reassignment until all nodes belonging to this branch of the ancestry tree have a new label (Steps 4-11).

4.2.4 Communication in BLWNet

Pairwise communication and single node broadcasting are the most common activities in a P2P network. In this section we show how the assigned labels can be used to facilitate communication between two nodes as well as efficiently allow a node to broadcast a message to the entire network.

Algorithm 13 partialRelabeling(B, c)

Input: BLWNet network $B(V_B, E_B)$, A node $c \in V_B$ initiation partial relabeling.
Output: B after partial relabeling.

```
1: List currentDonorList;
2: List nextDonorList;
3: currentDonorList.add(c);
4: repeat
5:    for each node v_i ∈ currentDonorList do
6:       Node v_i assign new label to each child(v_i);
7:       nextDonorList.add(child(v_i));
8:    end for
9:    currentDonorList.clear();
10:   currentDonorList.addAll(nextDonorList);
11: until nextDonorList.isEmpty()
12: return B
```

4.2.4.1 Routing

Assume node $v_i \in G$ wants to send a message to node $v_j \in G$. Furthermore, suppose that v_i knows v_j's label set $LS(v_j)$. If nodes v_i and v_j are neighbors the message can be relayed directly. However, two nodes belonging to a large P2P network are likely to be located multiple hops from each other. Each node in the path between v_i and v_j should be able to decide which one of their neighbors should be the next hop and forward the message based on this decision. The goal is to allow a node to make this decision based only on the destination label and its neighborhood information. So at the core of the routing process is the forwarding mechanism. In algorithm 14 we show the steps taken by a node when making a forwarding decision. Assume node *current* has just received a message from node $from$ with final destination node *to*. In the simplest case, if node *to* is one of *current*'s neighbors then the message is delivered and routing ends (Steps 6-7). Node *current* would then confirm message delivery to node $from$ which in turn would relay the confirmation back at the originating node using the same recursive routing path. In the most likely event node *to* is not a neighbor, node *current* would create a selected list of neighbors as possible next hop choices. This list does not include node $from$ or

any other neighbor who has already received this message (Step 9). If no such choices are available, node *current* would reply back to node $from$ informing that the message cannot be forwarded by *current*, in which case $from$ has to find an alternative path to destination (Steps 11-13). Otherwise, *current* would find the best choice from its list by using as criteria the label match between each candidate and the destination node. Ties are broken by using node dimension and finally node ID (Steps 14-24). If the chosen node *next* has a better match than the current node, the message would be forwarded to node *next*. Otherwise the message is forwarded to the parent of the current node (Steps 25-31). If the choice of node *next* does not work, then *current* would test the other choices if any are available. The backtracking mechanism employed in this forwarding algorithm assures that a route is identified if the originating and destination nodes are connected.

Finally, once the forwarding procedure is identified, the routing algorithm is simple. Algorithm 15 serves as the initiation of the first forwarding step in identifying the route to destination. Routing examples in BLWNet networks can be found in section 4.2.5.2 and appendices C and D.

4.2.4.2 Broadcasting

To broadcast a message to the entire network, a node makes use of the ancestry tree. Details of the broadcasting procedure are shown in Algorithms 16 and 17.

Let B represents a BLWNet instance. Assume node $v_0 \in B$ wants to send a message to all other nodes in B. Node v_0 sends the message to its parent and all $child(v_i)$ if any. Exception is made when v_0 is the ancestry tree root in which case the message is only forwarded to its children (See Algorithm 16). Recursively, if a node v_i receives the broadcast message from its parent, it forwards the message to all $child(v_i)$ nodes if any. Otherwise, if v_i receives the message from one of its children nodes, it forwards

Algorithm 14 forwardMessage(B, current, from, to)

Input: BLWNet Network B, Current Node $current \in V_B$, Previous Node $from \in V_B$, Destination Node $to \in V_B$.
Output: True if message is delivered, False otherwise.

```
1: BOOLEAN routingWorks;
2: LIST neighbors;
3: NODE next;
4: INT thisNodeMatch ← labelMatch(current, to);
5: INT bestMatchFound ← 0;
6: if current.equals(to) then
7:   return true;
8: else
9:   neighbors ← v|v ∈ neighborhood(current), !v.equals(from), !v.isVisited();
10:  repeat
11:    if neighbors.size() == 0 then
12:      return false;
13:    end if
14:    next ← neighbors.get(0);
15:    bestMatchFound ← labelMatch(next, to);
16:    for each Node v_i ∈ neighbors do
17:      if labelMatch(v_i, to) > bestMatchFound then
18:        next ← v_i;
19:        bestMatchFound ← labelMatch(v_i, to);
20:      else if labelMatch(v_i, to) == bestMatchFound&&dimension(v_i) <
         dimension(next) then
21:        next ← v_i;
22:        bestMatchFound ← labelMatch(v_i, to);
23:      end if
24:    end for
25:    if bestMatchFound > thisNodeMatch then
26:      routingWorks ← forwardMessage(next, current, to);
27:    else if !current.getParent().isVisited() then
28:      routingWorks ← forwardMessage(current.getParent(), current, to);
29:    else
30:      routingWorks ← forwardMessage(next, current, to);
31:    end if
32:    if !routingWorks then
33:      neighbors.remove(next);
34:    end if
35:  until !routingWorks
36: end if
37: return true;
```

Algorithm 15 BLWNetRouting(B, s, d)

Input: BLWNet Network B, Source Node $s \in V_B$, Destination Node $d \in V_B$.
Output: A path P from s to d, where $P \subseteq E_B$.

```
1: forwardMessage(B, s, s, d)
2: return
```

the message to its parent and all other $child(v_i)$ nodes if any (See Algorithm 17). The message is recursively forwarded until all nodes are reached.

Algorithm 16 BLWNetBroadcasting(B, s)

Input: BLWNet Network B, Source Node $s \in V_B$.
Output: A complete broadcast tree with root at node s.

```
1: if parent(s) != null then
2:    forwardBroadcast(B, parent(s), s);
3: end if
4: for each child_i(s) do
5:    forwardBroadcast(B, child_i(s), s);
6: end for
7: return
```

Algorithm 17 forwardBroadcast(B, current, from)

Input: BLWNet Network B, Current Node $current$, Previous Node $from$.
Output: Message is forwarded to a subset of nodes from B.

```
1: if from.equals(parent(current)) then
2:    for each child_i(current) do
3:       forwardBroadcast(B, child_i(current), current);
4:    end for
5: else
6:    if parent(current) != null then
7:       forwardBroadcast(B, parent(current), current);
8:    end if
9:    for each child_i(current) except node from do
10:      forwardBroadcast(B, child_i(current), current);
11:   end for
12: end if
13: return
```

Figure 4.8 shown an example of a broadcast in a BLWNet of 7 nodes. Subfigure 4.8(a) shows the ancestry tree for the network and subfigure 4.8(b) shows a broadcast from

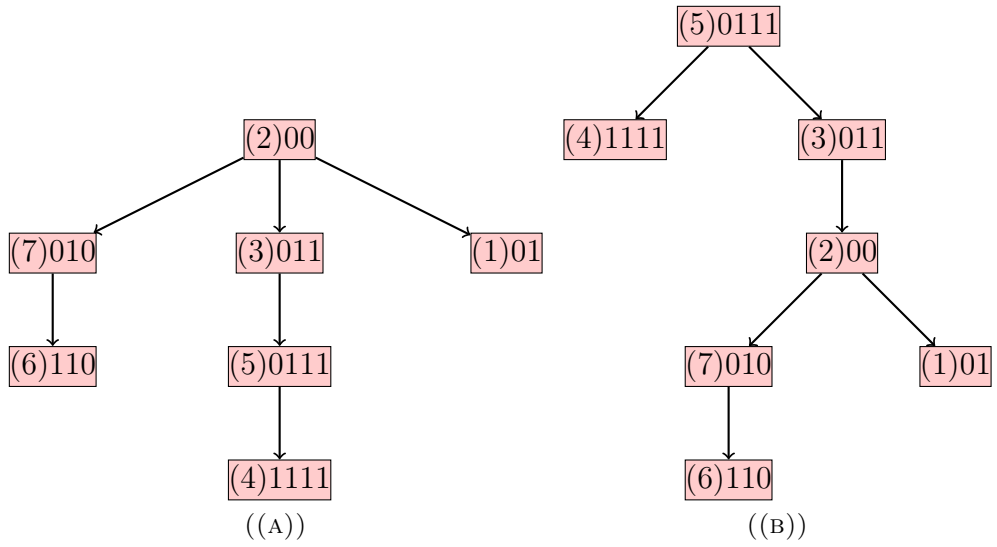

FIGURE 4.8: Broadcast Example in BLWNet

node (5) using the ancestry tree. Additional examples can be found in section 4.2.5.2 and appendices C and D.

4.2.5 Experimental Evaluation

We conduct a number of experiments to illustrate and test the properties of BLWNet networks. The goal of these experiments is to: (1) construct and visualize random instances of a BLWNet network; (2) simulate and monitor label distribution during the network deployment as well as changes in label assignment following network changes; (3) show that the distribution is done correctly and is feasible for large networks, and correlate the growth of the label size to the growth of the network size; (4) monitor the correct deployment and maintenance of the parent-child relationships and the ability to maintain the ancestry tree; (5) simulate network activities such as pairwise routing and broadcasting using the assigned labels; (6) assess the efficiency of the routing algorithm

by comparing the resulting average path length to the optimal path length of a BLWNet instance; (7) perform a global broadcast, monitor to ensure that all nodes are reached, and estimate the complexity of such broadcasts.

4.2.5.1 Experimental Setup

To create an instance of a BLWNet and test all algorithms described in this chapter we implemented a JAVA program using the Eclipse platform. Results such as the set of nodes, connections between nodes, routing and broadcasting information are printed in files. These files are then visualized using Cytoscape [74, 75] version 2.8.3. Cytoscape is a software tool for visualizing networks. This software specializes in visualizing genomic interaction networks but is well suited for all types of networks. Cytoscape offers a user friendly interface, very neat and appealing visual displays, cross platform compatibility and detailed network statistics.

A number of BLWNet networks are randomly generated. These networks vary in size from 16 to 1024 nodes. We chose the number of nodes to be powers of two for a better and clearer plotting of results on a log scale. To generate these networks a number of parameters are used. Nodes are deployed in a 2-dimensional square area. Each node is randomly assigned two coordinates determining the location of the node within the deployment area. Nodes are also randomly assigned a range from an interval of values. Once all node locations and ranges are identified a link is created between two nodes if they are within range of each other, meaning they can both transmit and receive a message. Thus the final result is an undirected graph. The resulting network is then tested for connectivity. Network connectivity is one of the assumed properties of the network so that all random instances which are not connected are discarded. Based on the choice for the size of the area, number of nodes and node range, we obtain a large variety of random networks both sparse and dense.

4.2.5.2 Experimental Results

We start with a BLWNet example of 32 nodes. The resulting graph, representing this random BLWNet instance, is shown in figure 4.9. The size and gradient color for each node is designed to reflect the number of connections the node maintains. The label set for each node is also displayed. To show the label distribution steps we print and display the corresponding ancestry tree for the network (figure 4.10). The root node (node (8)) from the tree is the first selected node to start the label distribution. All paths to the tree leaves show the steps that are taken to split the label set among all nodes.

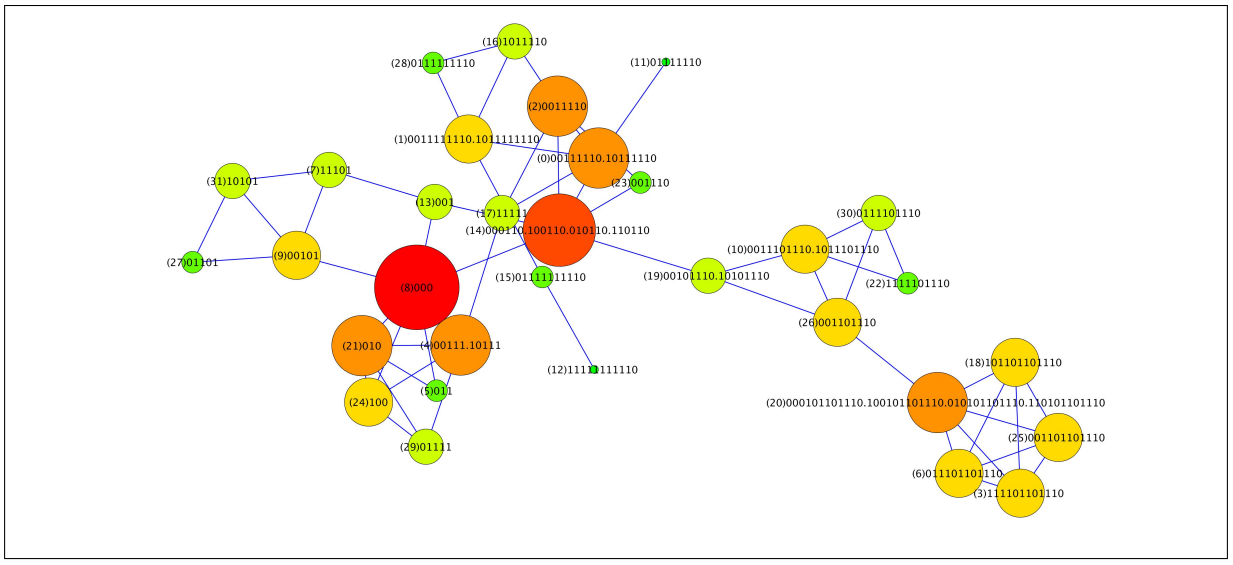

FIGURE 4.9: BLWNet Example with 32 Nodes

Figure 4.11 shows the node degree distribution. As expected, the distribution is approximately bell-shaped, a direct consequence of the random nature of the network.

As a BLWNet grows in size, so does the length of each binary label. We look at the label growth pattern in terms of the size of the network. Figure 4.12 shows the results obtained by testing a number of BLWNet networks of various sizes ranging from 16 to 1028. We take the size of each network instance to be a power of two for an easier display on a log scale. We observe that the label size tends to grow logarithmically compared with the

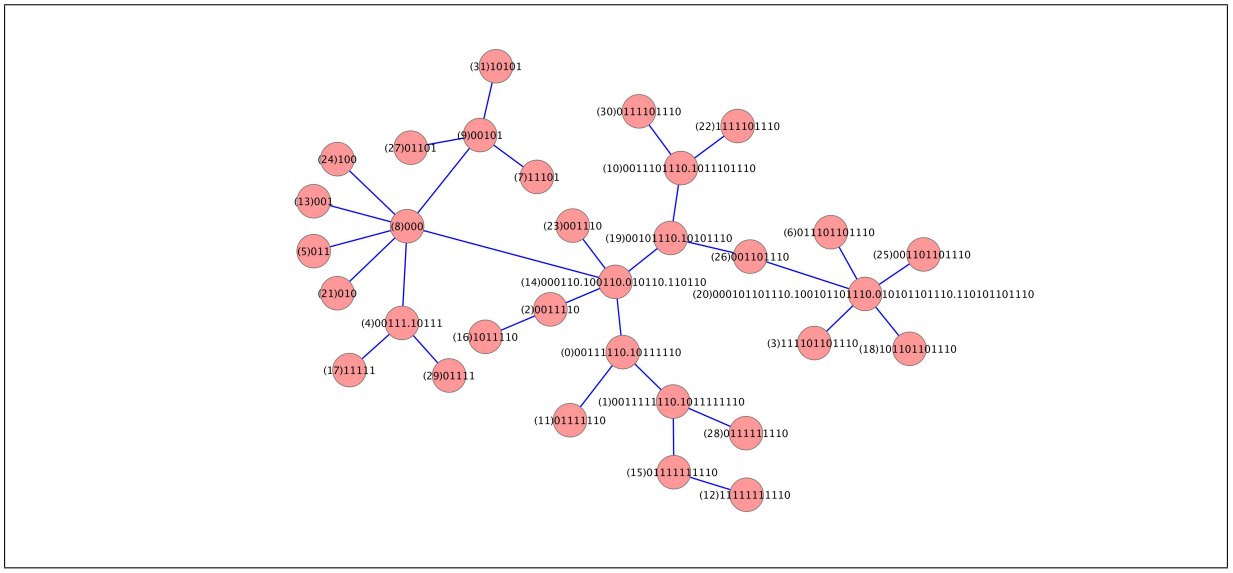

FIGURE 4.10: Ancestry Tree for the Network in Figure 4.9

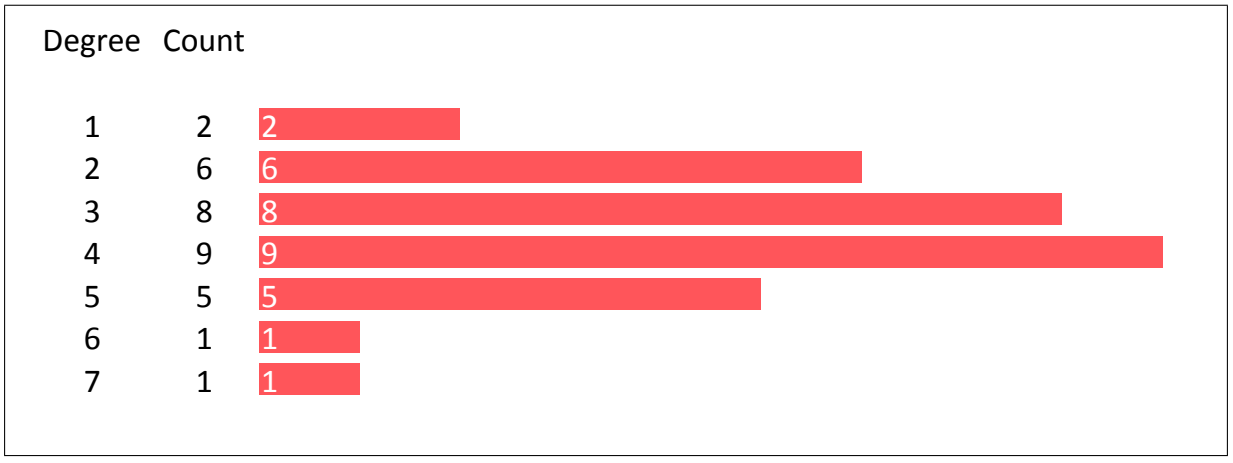

FIGURE 4.11: Node Degree Distribution for the Network in Figure 4.9

number of nodes. There is clearly a direct relation between label size and the centrality of the most connected node. As a consequence there is a direct correlation between the network diameter and the size of the largest label.

We then test the BLWNet routing algorithm. The most important observation is the fact that a route is always identified between two nodes only using the assigned labels and any local information. To illustrate the nature of typical routes we show here some examples using a BLWNet network of 64 nodes (figure 4.13). Three routing path examples are shown in figures 4.14-4.16.

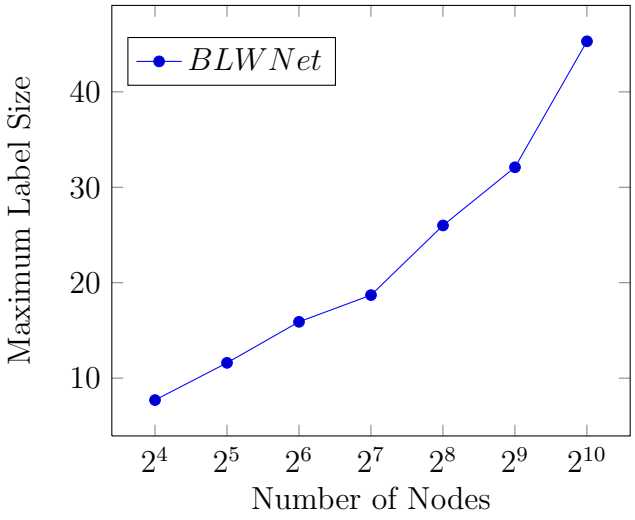

FIGURE 4.12: Growth of Maximum Label Size with BLWNet Network Size

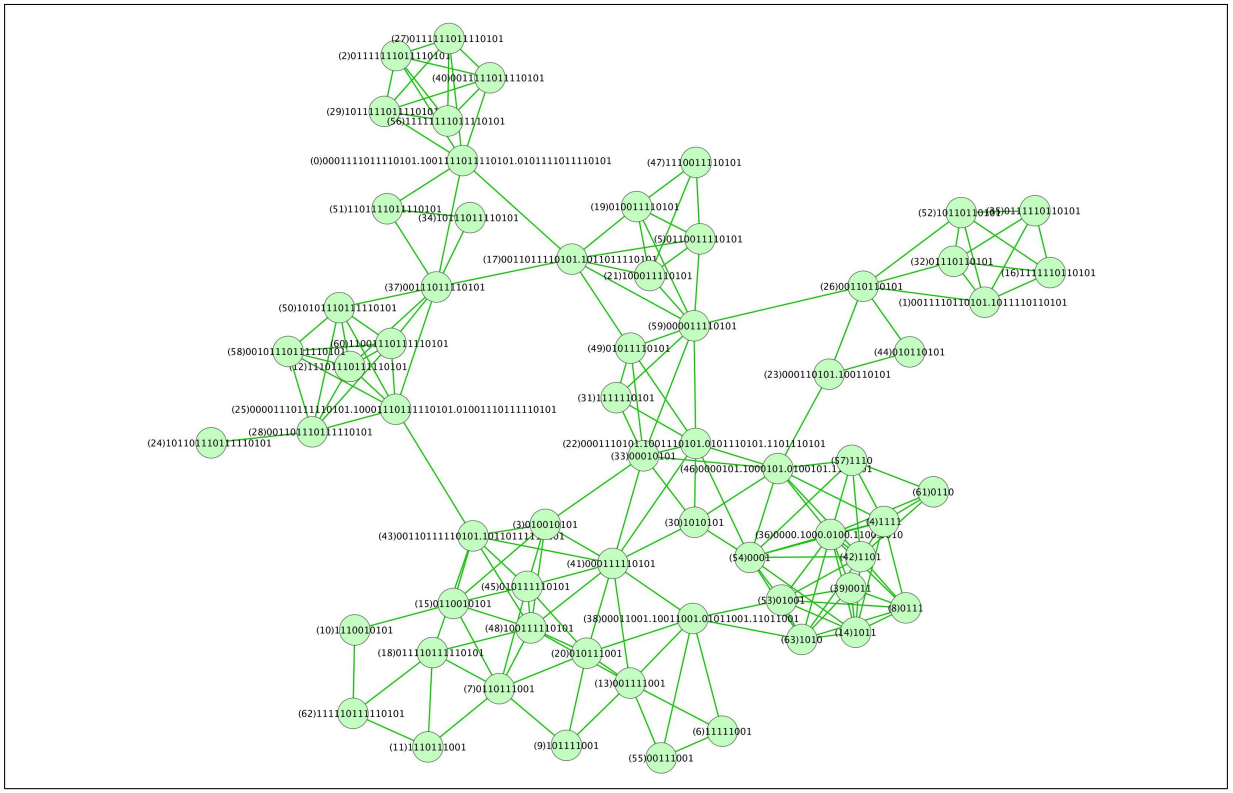

FIGURE 4.13: BLWNet Instance of 64 Nodes Used to Show Routing Examples

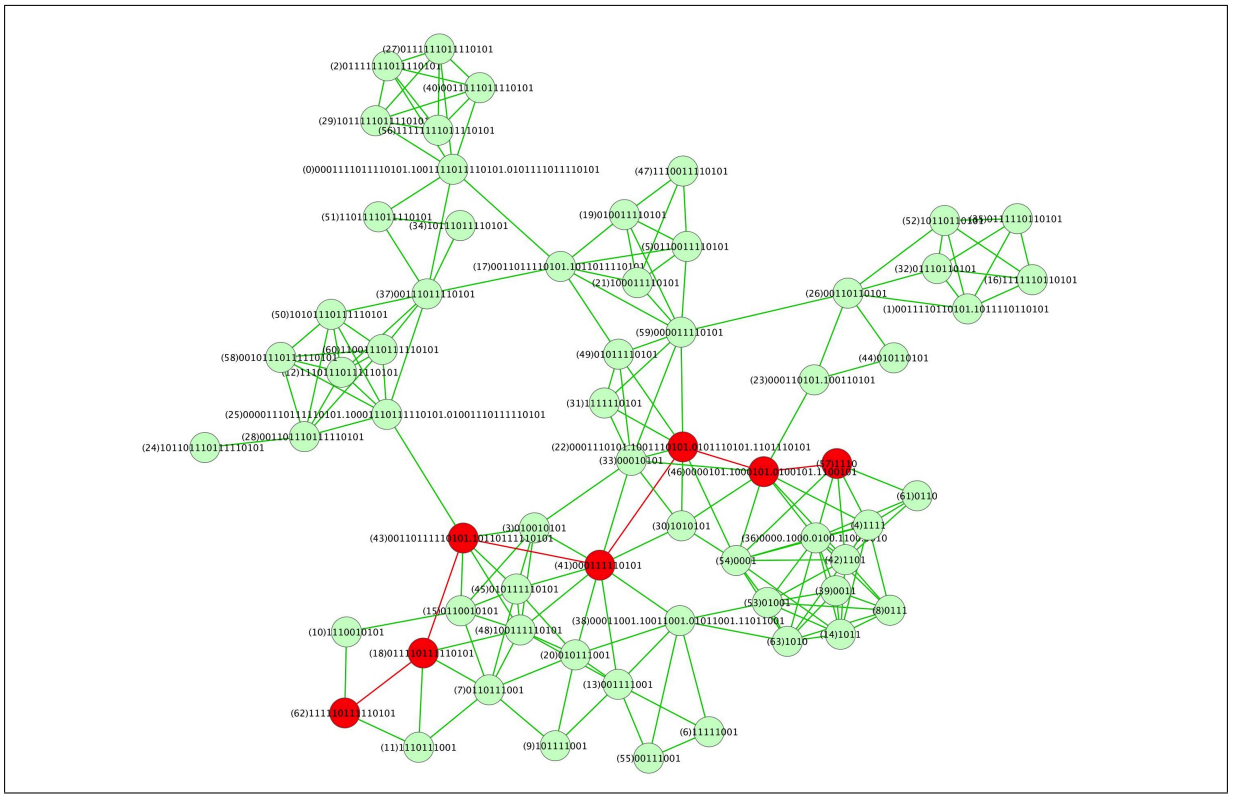

FIGURE 4.14: Routing Example From Node 62 to Node 57

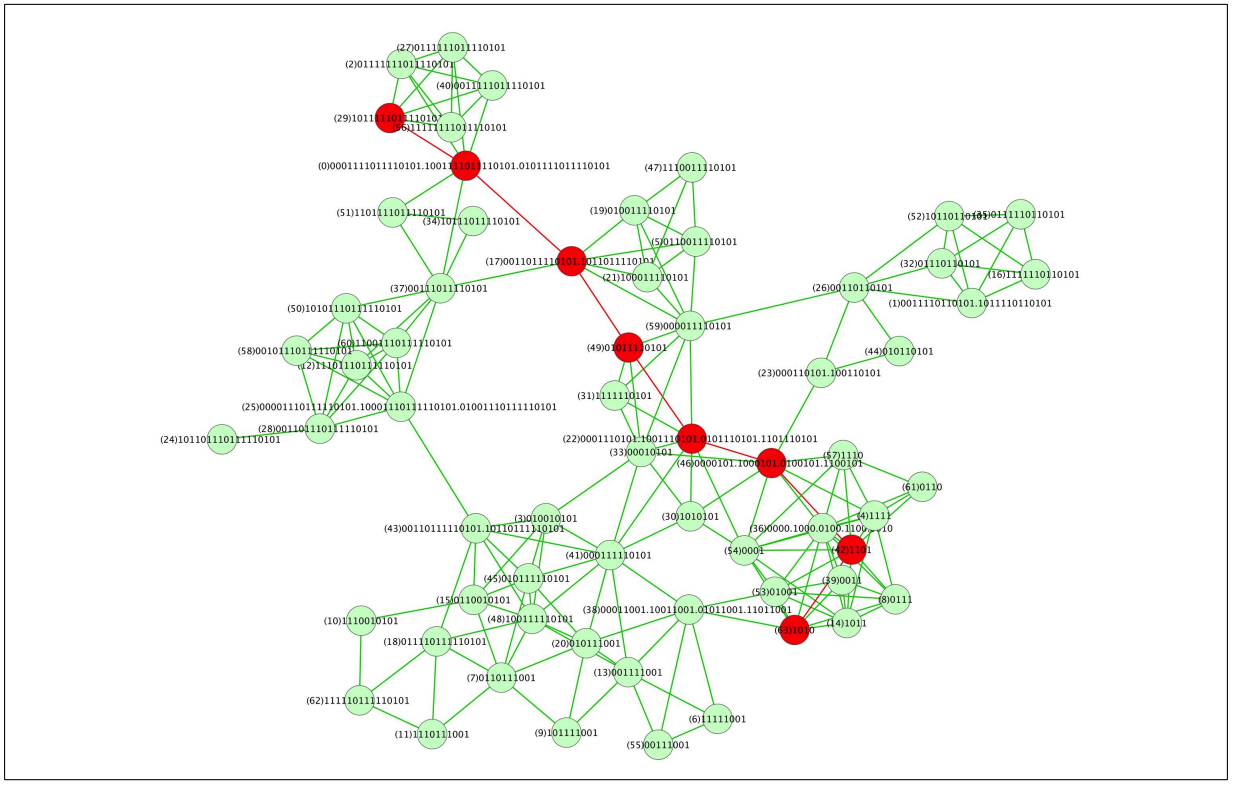

FIGURE 4.15: Routing Example From Node 63 to Node 2

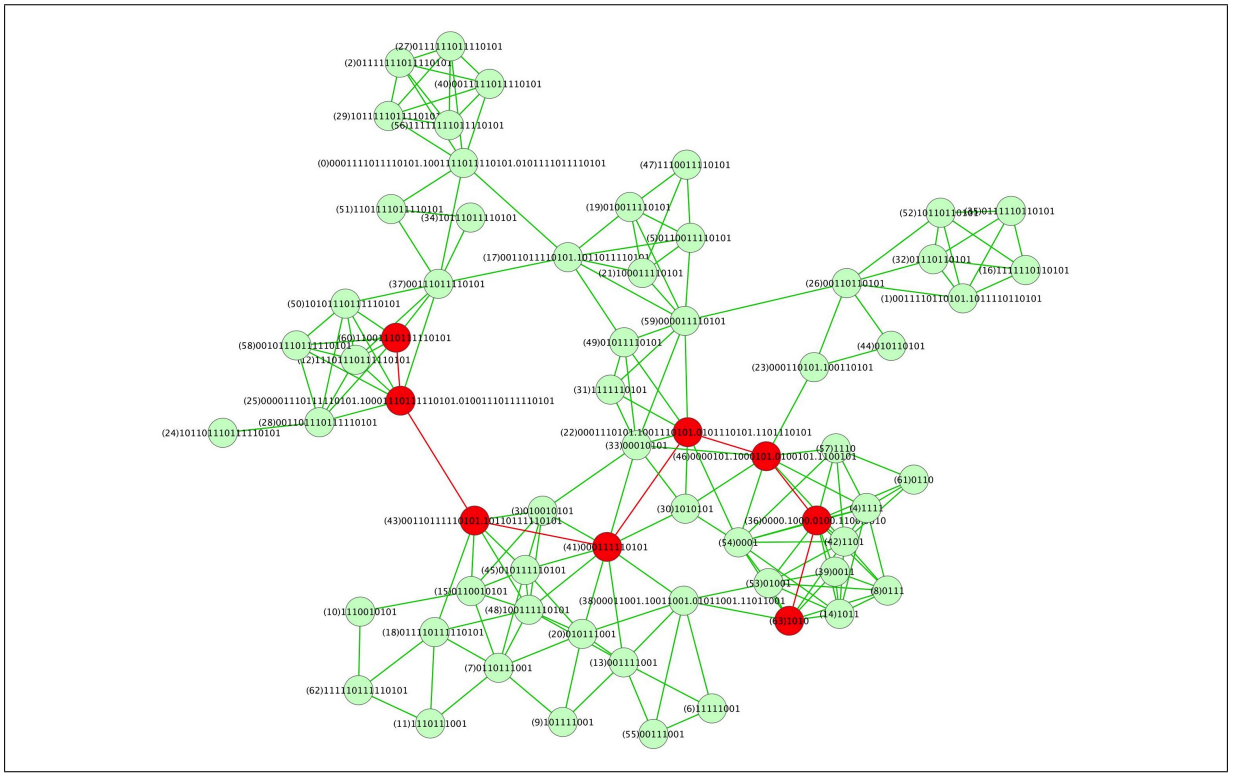

FIGURE 4.16: Routing Example From Node 60 to Node 63

We choose a larger BLWNet instance of 128 nodes (figure 4.17) to illustrate the differences between the BLWNet path length distribution and the optimal path length distribution. We apply the BLWNet routing algorithm to identify a route between any pair of nodes in the network. The observed results are displayed in the bar chart in figure 4.18. A side by side comparison of path length frequency is done with the bar chart summarizing the optimal path lengths. The optimal path length distribution and statistics are obtained using the built in features of the Cytoscape software.

For a more accurate assessment of the performance of the BLWNet routing algorithm we conduct more extensive simulations with networks of various sizes. We compare the average observed BLWNet paths to average optimal paths. Separate comparisons are done for different BLWNet variations. First we look at the performance of BLWNet immediately after the initial deployment and before any topology changes take place (see figure 4.19). We then test BLWNet instances following a significant number of

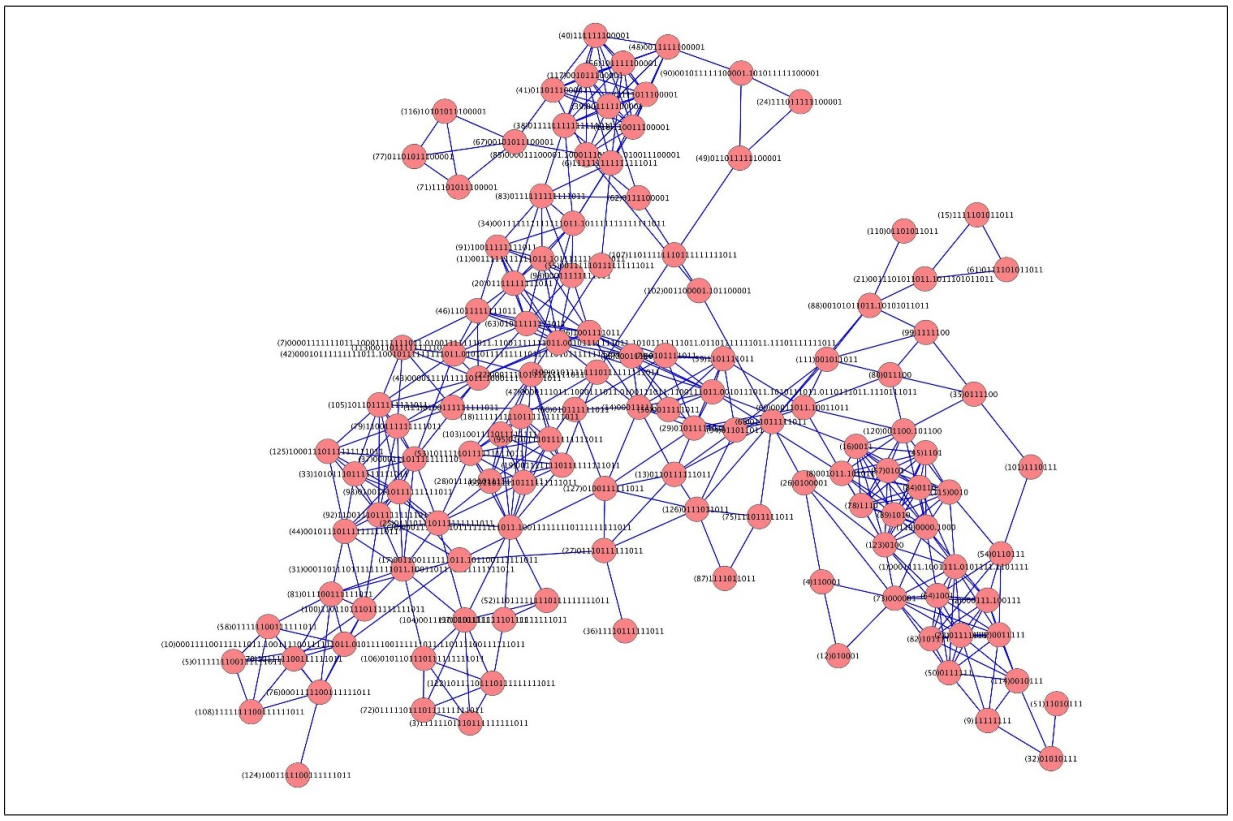

FIGURE 4.17: BLWNet of 128 Nodes Used for Path Length Distribution Comparison

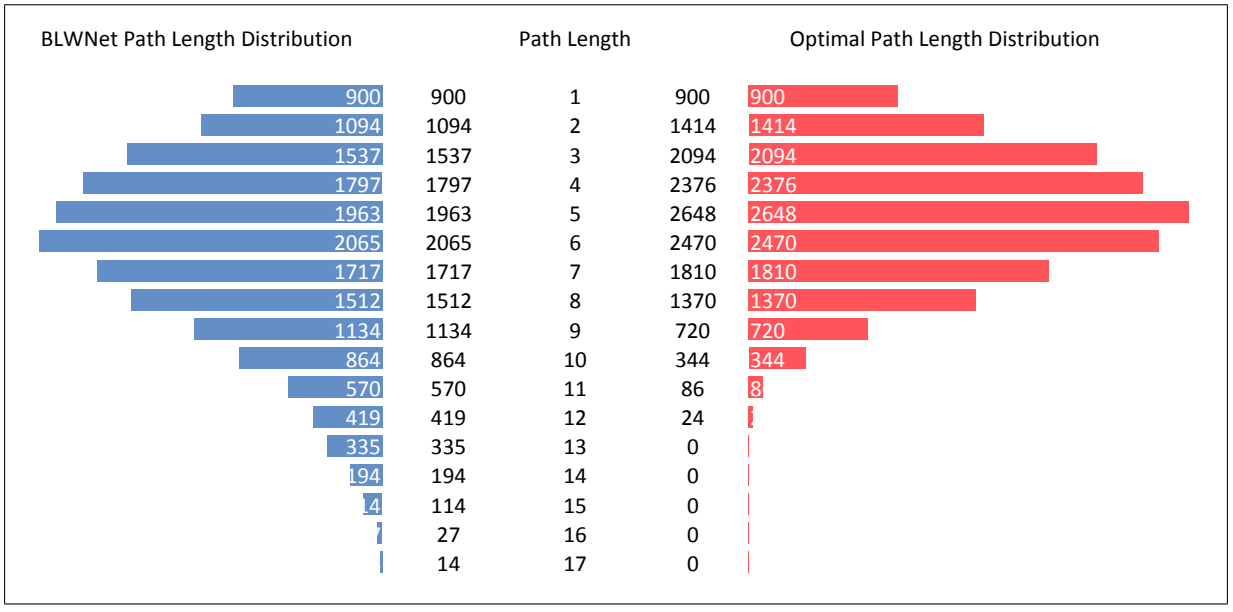

FIGURE 4.18: Path Length Distribution Comparison for the Network in Figure 4.17

network changes such as node enters and node deletions (see figure 4.20). Lastly we observe the efficiency of the routing algorithm following network changes and applying the relabeling procedure when parent-child connections change following the deletion of a node. All results are then combined in a single graph shown in figure 4.22. In every case, we can observe that the routing algorithm performs quite well in every type of network. The performance however decreases following a significant number of network changes. Typically, small improvements are observed when employing a partial relabeling adjustment. Finally, we can conclude that during the existence of BLWNet network, following a large number of network changes, the performance of the routing algorithm decreases. In such cases, it may be beneficial to redeploy the BLWNet and introduce a fresh label assignment.

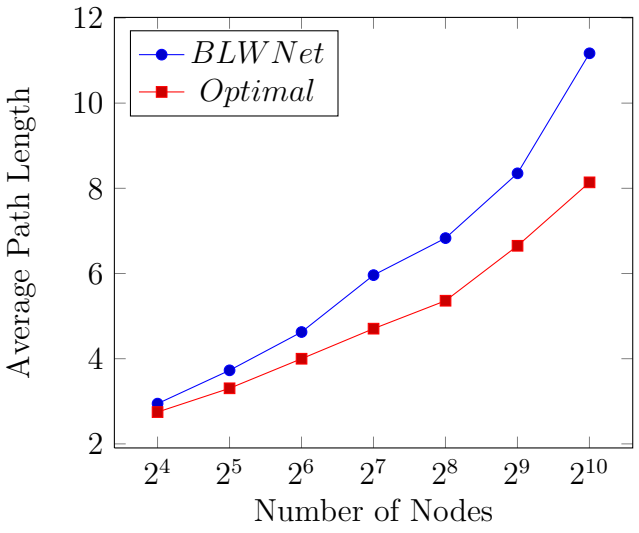

FIGURE 4.19: Average Path Length Comparison No Topology Changes

4.2.6 Summary

In this chapter we proposed a labeling schema for ad-hoc wireless networks. The proposed protocol is shown to improve communication in a number of ways. The labeling schema does not create an overlay so participating nodes connect and communicate directly with

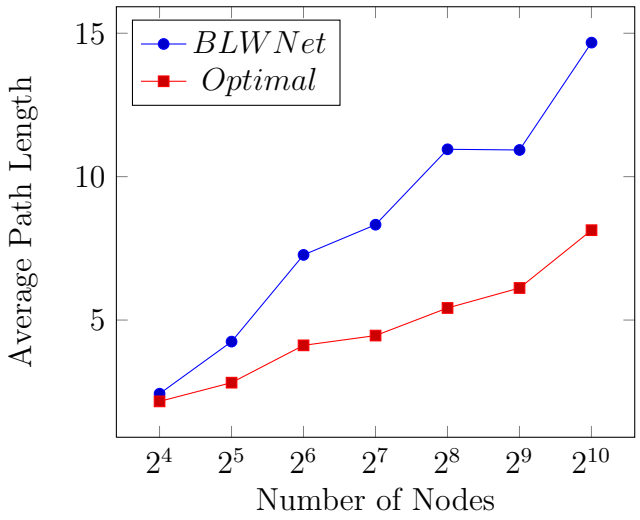

FIGURE 4.20: Average Path Length Comparison With Topology Changes

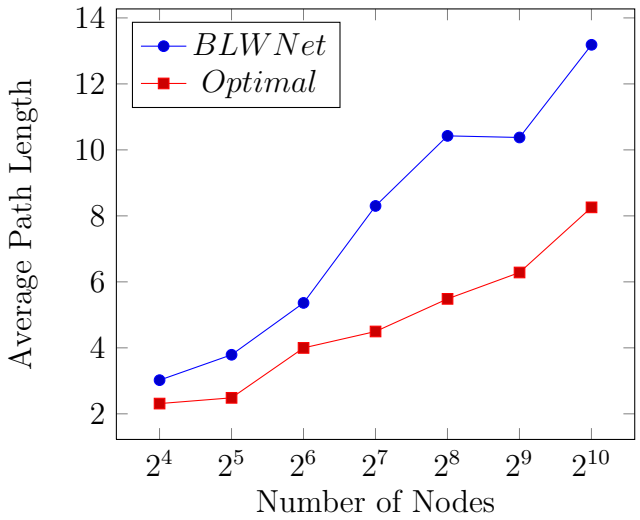

FIGURE 4.21: Average Path Length Comparison With Topology Changes and Partial Relabeling

their physical neighbors only. An efficient next hop decision-making process facilitated by the use of the assigned labels allows nodes to adjust their transmission range dynamically so that only the destination neighbor is reached instead of using the maximum power to reach all neighbors. Only knowledge of the corresponding label sets are necessary for two nodes to be able to communicate with each other. A path between them is dynamically selected and there is no need to maintain large routing tables which are commonly used in ad-hoc networks. In addition, it is possible to identify alternative routes in case of node

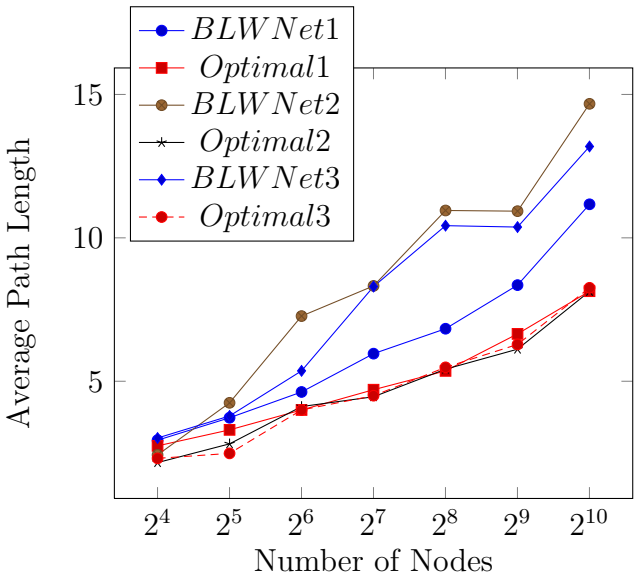

FIGURE 4.22: Average Path Length Comparison Combined

failure or sudden changes in the topology. This is particularly important considering that the network may be subject to frequent changes over time. Broadcasting to the entire network can also be performed efficiently, avoiding costly approaches such as flood queries. As a result of the reduced number of messages needed during routing and the adaptive transmission range at which each node operates, the interference is likely to decrease. Reduced interference in turn will reduce the number of retransmissions and increase the quality and reliability of communication.

All these positive properties of the new approach have an associated cost in terms of establishing and maintaining the label sets and the relationships between neighboring nodes. This overhead is however expected to be quite small compared to potential savings. Furthermore the network is scalable and deals well with network changes since all node additions and deletions are performed locally.

Chapter 5

Conclusions

5.1 Summary

This thesis introduces two labeling schemata for dynamic P2P networks. We show detailed procedures for deployment and maintenance of these engineered networks. We also show how the assigned labels can be used to increase the efficiency of communication. We prove different properties of these networks via theoretical discussions and extensive experiments. Here we list the most important observations and conclusions: (1) Label assignment is a feasible technique in dynamic distributed P2P networks. (2) The assigned labels can improve the way nodes form connections. These connections can be arranged to form a structured overlay approximating known graphs such as hypercubes. (3) Labels can also be used to make good routing and forwarding decisions. As a result routing and broadcasting is simpler and more efficient. Alternative routes are much easier to identify following any network changes or node failures. (4) Label assignment and reassignment is only done locally, thus impacting a small subset of nodes and keeping the associated cost to a minimum. (5) Labels can be useful even if a logical overlay is not formed,

where physical nodes communicate using their physical connections, thus eliminating the negative effects of possible network misalignment.

In conclusion, structured labeling offers a number of advantages in a large variety of distributed networks. However, there is a maintenance cost associated with label distribution and maintenance. This cost increases in highly dynamic networks where changes in topology are quite frequent. It is important to identify those conditions in which a hypercube topology performs better than a random, unstructured network. To estimate these conditions we have previously looked at a simple average cost comparison models for identifying trade-offs in hypercube networks [66]. We use representative average values to compare query execution costs in a structured hypercube network and a scale free network that grows based on a preferential attachment protocol.

Let HC be a hypercube network and PA be a preferential attachment scale free network. Let $Cost(N)$ be the total bandwidth utilization per minute for network N. We estimate that the total bandwidth utilization can be approximated by the formula $Cost(N) = CQ \times RQ + CC \times RC + BC$, where RC is the rate of topology changes (node enters and deletions per minute), CC is the average estimated cost of a single topology change (in kb), RQ is the rate of queries issued each minute, CQ is the average cost of a single query (in kb) and BC is the background cost for periodically checking and maintaining network connections. We then select representative values for each of these parameters, based on experimental results using small HC and PA instances. We obtain the following approximate bandwidth utilization formulas for each test instance:

$COST(HC) = 35 \times RQ + 300 \times RC + 855.$

$COST(PA) = 165 \times RQ + 100 \times RC + 200.$

We plot the resulting formulas in 3-dimensional space, assigning various representative values to RC and RQ and calculating the corresponding $COST(HC)$ and $COST(PA)$.

The results are shown in Figure 5.1. It can be seen that the hypercube network performs better when the rate of queries increases and the scale free network is a better choice when the rate of changes is significantly higher than the rate of queries issued. Similar results can be obtained using different representative values and more parameters. In conclusion, given the large variety of P2P networks that are different in scope and characteristics, it is unlikely to design a protocol whose performance is superior for each P2P instance. Trade-offs exist and more specific protocols need to be designed to satisfy the specific needs of a P2P network. This work offers a clear direction for improving performance while minimizing associated costs.

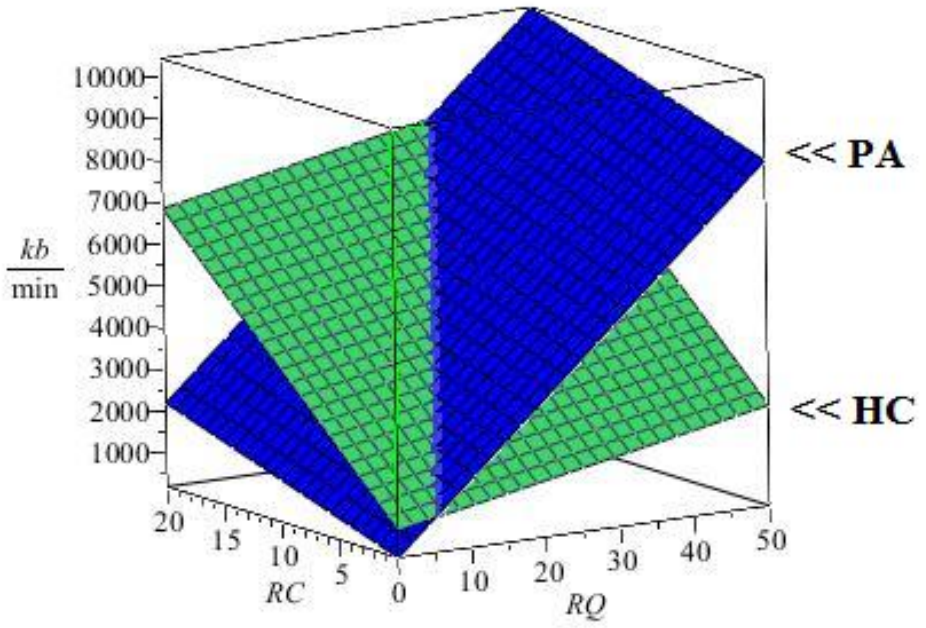

FIGURE 5.1: 3D Bandwidth Utilization Comparison Graph for Selected Topologies

5.2 Future Work

Possible directions for future work include: (1) Dealing with highly dynamic networks where multiple nodes may enter of be deleted simultaneously. (2) Identify different ways of increasing the alignment of the logical overlay to the corresponding physical network. (3) Identify efficient ways of merging or splitting structured P2P networks while maintaining the properties of the intended engineered network. (4) Consider a periodical label redistribution, minimizing the probability that some nodes accumulate larger label sets following continuous network changes. Assess the benefits of a more uniform label distribution. (5) Identify possible uses of the assigned labels to decrease the vulnerability of each network and increase network security. (6) Use the assigned labels to increase the efficiency of more complex network activities such as query joins and semi-joins. (7) Allow nodes belonging to the same overlay to function at different hypercube dimensions so that the resulting logical network resembles more the underlying physical substrate. (8) Consider other known graphs as a model for the logical overlay. Choices may include circulant, Kautz and De Bruijn networks.

Appendix A

HyperD Examples

A.1 Examples of Network Changes

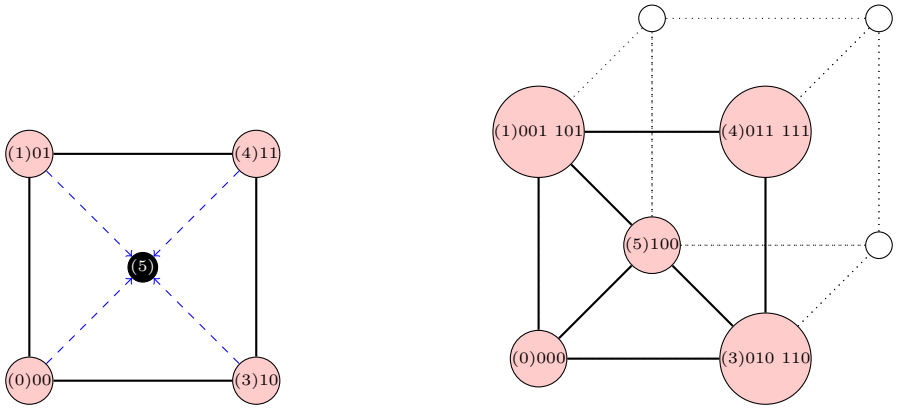

FIGURE A.1: Example of a Node Entering a Full 2-d HyperD

Figure A.1 shows an example of a node entering a full 2-dimensional HyperD. Node (5) broadcasts an enter request. Since the network is a full hypercube (figure A.1(a)), it needs to expand to a 3D hypercube. 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 A.1(b).

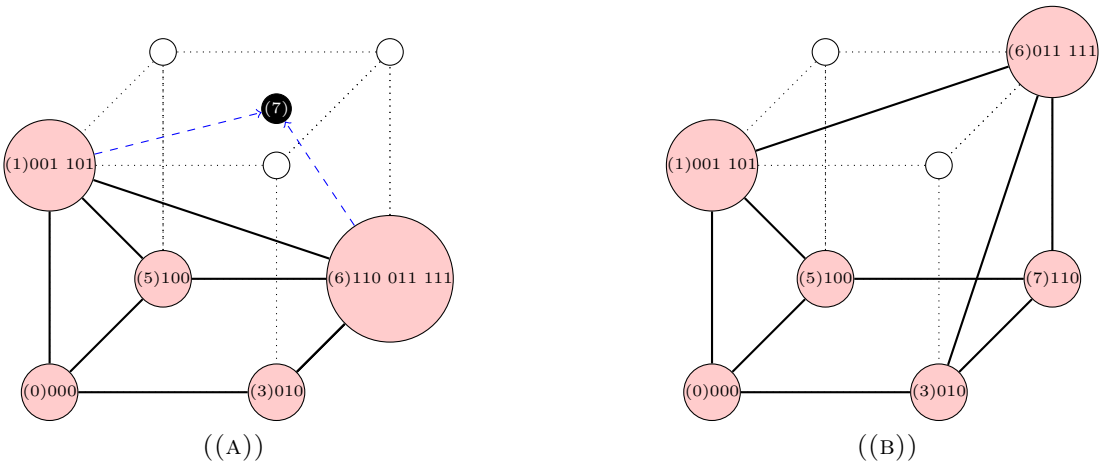

FIGURE A.2: Example of a Node Entering a Partial 3-d HyperD

Figure A.2 shows an example of a node entering a partial 3D HyperD. Node (7) broadcasts a enter request and receives a response from nodes (1) and (6) which have available labels in their label space (figure A.2(a)). Assume node (7) accepts a label from node (6) . (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 A.2(b).

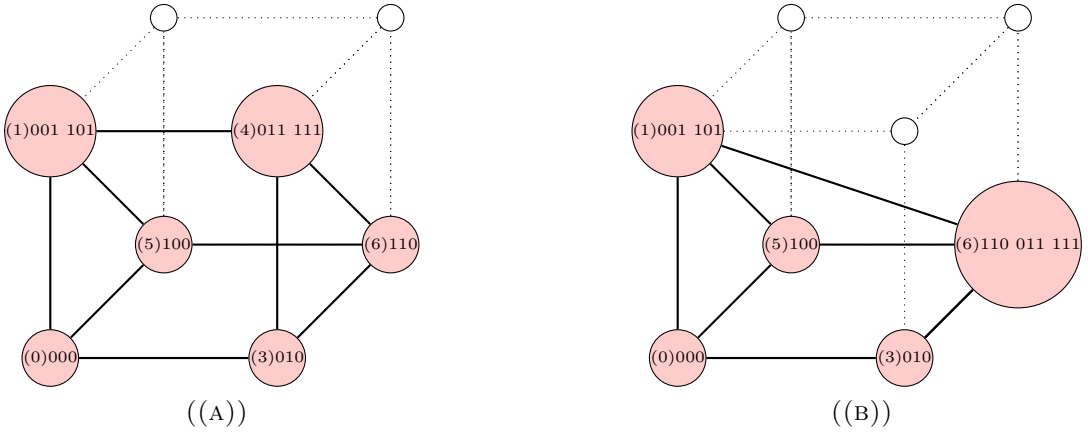

FIGURE A.3: Example of a Node Departing a HyperD

Figure A.3 shows an example of a node departing HyperD. Node (4) leaves the HyperD network shown in figure A.3(a). Node (6) assumes control of the label space 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 A.3(b).

A.2 13 Node HyperD Deployment and Maintenance Example

Figures A.4 and A.5 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:

- (u)LS(u) is node u in HyperD with label set $LS(u)$.
- (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 A.4(a): HyperD consists of a single node (0) .
- Part A.4(b): Node (1) wants to enter. HyperD expands to the 1st dimension.

- Part A.4(c): ❶ connects to ⓪ to form a full 1-dimensional hypercube.
- Part A.4(d): Node ❷ wants to enter. HyperD expands to the 2nd dimension since there no available labels.
- Part A.4(e): Node ❷ Get label 11 from node ① which is the first node to respond to the enter request. Node ❷ connects to ① and ⓪ .
- Part A.4(f): Node ❸ wants to enter. Node ⓪ is the only node to respond.
- Part A.4(g): Node ❸ gets label 10 and fills the last position in the hypercube.
- Part A.4(h): Node ② departs. Node ③ takes over the label from ② since it is the lowest dimension neighbor. ③ connects to ① .
- Part A.4(i): Node ❹ wants to enter. Node ③ is the only node to respond.
- Part A.4(j): Node ❹ gets label 11 and fills the last position in the hypercube.
- Part A.4(k): Node ❺ wants to enter. HyperD expands to the 3rd dimension since there no available labels.
- Part A.4(l): Node ❺ gets the largest label 100 from the first responding node ⓪ . ❺ connects to ⓪ , ① and ③ .
- Part A.4(m): Node ❻ wants to enter. Node ③ is the first to respond.
- Part A.4(n): Node ❻ gets the largest label 110 from ③ and connects to ③ , ④ and ⑤ .
- Part A.4(o): Node ④ departs. Node ⑥ takes over the label set since it is the lowest dimension neighbor relative to the largest label 111 from ④ . ⑥ connects to ① .
- Part A.4(p): Node ❼ wants to enter. Node ⑥ responds first.

- Part A.4(q): Node (6) donates label 110 which is the largest label with internal degree 1. (7) connects to (3) , (5) and (6) . Node (6) disconnects from (5) .
- Part A.4(r): Node (8) wants to enter. Node (1) responds first.
- Part A.4(s): Node (8) gets the largest label 101 from (1) and connects to (1) , (5) and (6) .
- Part A.4(t): Node (9) wants to enter. Node (6) is the only one to respond.
- Part A.4(u): Node (9) gets the largest label 111 and fills the last vacant position in the 3-dimensional hypercube.
- Part A.5(a): Node (10) 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 (9) which donates its largest label 1111. Node (10) then connects to nodes (6) , (7) , (8) and (9) .
- Part A.5(b): The new node (11) gets label 1001 from node (1) and connects to nodes (0) , (1) , (6) and (8) .
- Part A.5(c): The new node (12) gets label 1101 from node (8) and connects to nodes (5) , (8) , (10) and (11) .
- Part A.5(d): Node (7) departs. Node (10) takes control of all labels since its the smallest dimension neighbor relative to (7) 's largest label 1110. Node (10) connects to (5) .

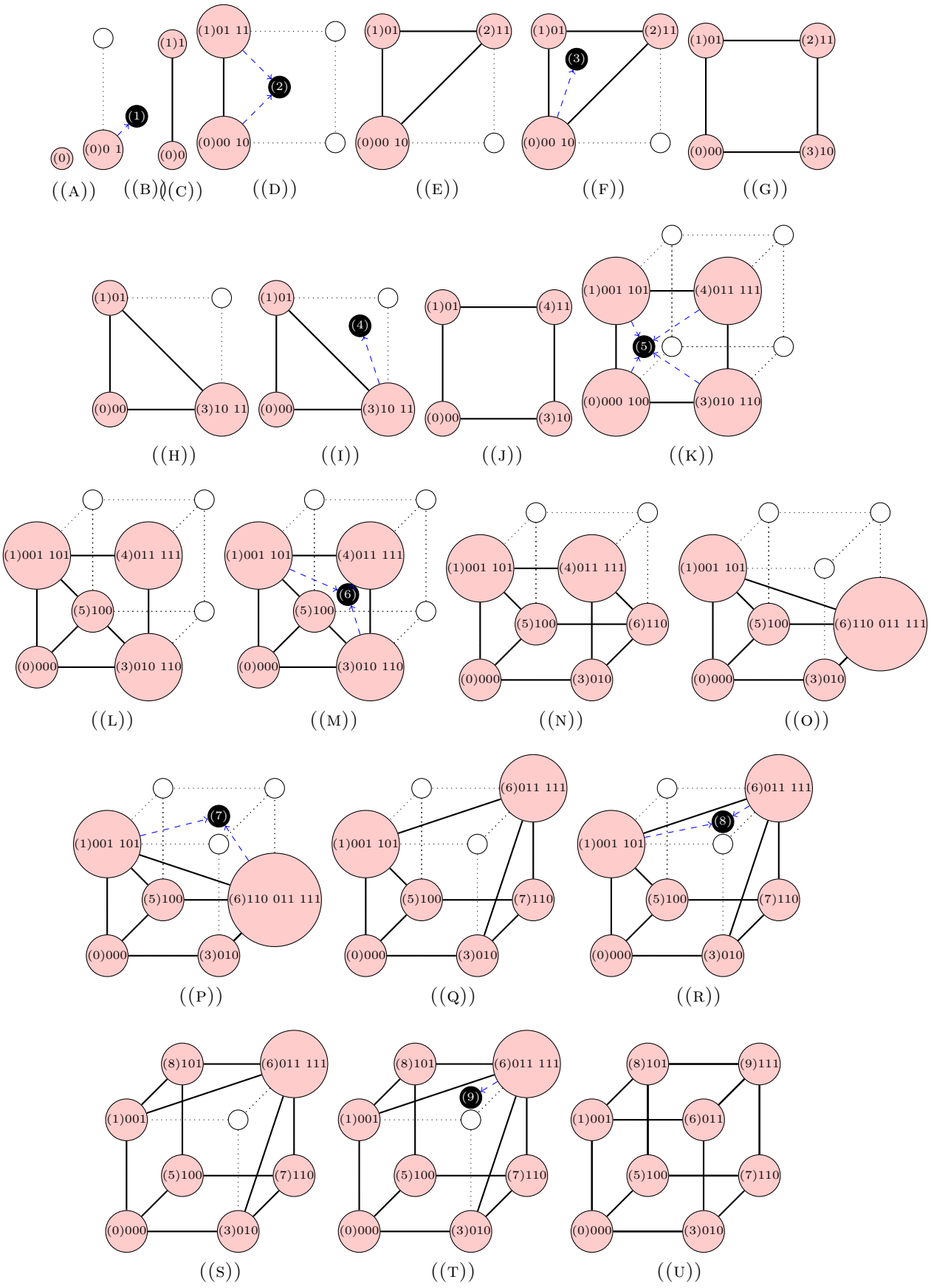

FIGURE A.4: HyperD Growth Example Part 1

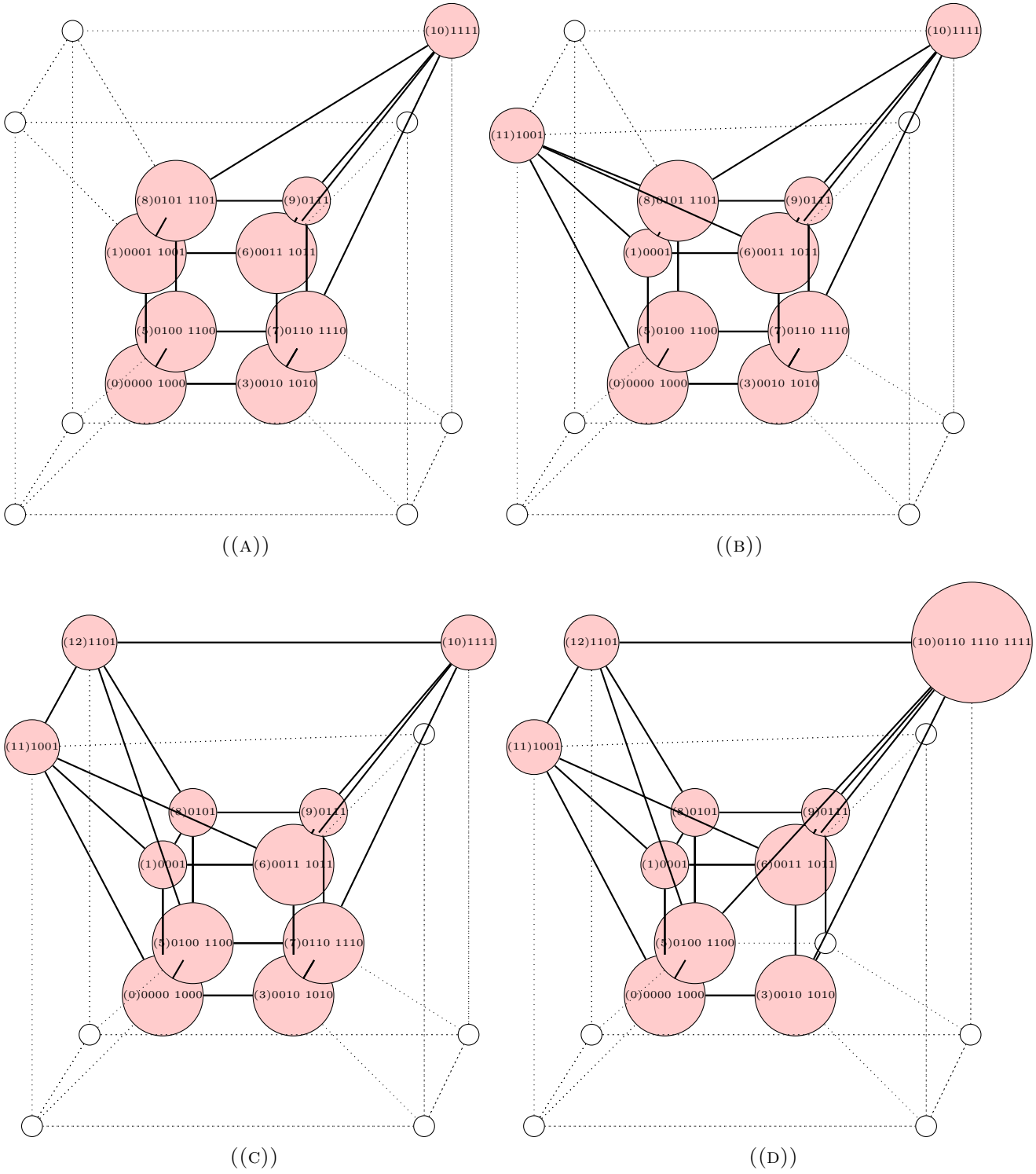

FIGURE A.5: HyperD Growth Example Part 2

Appendix B

HyperD Experimental Results

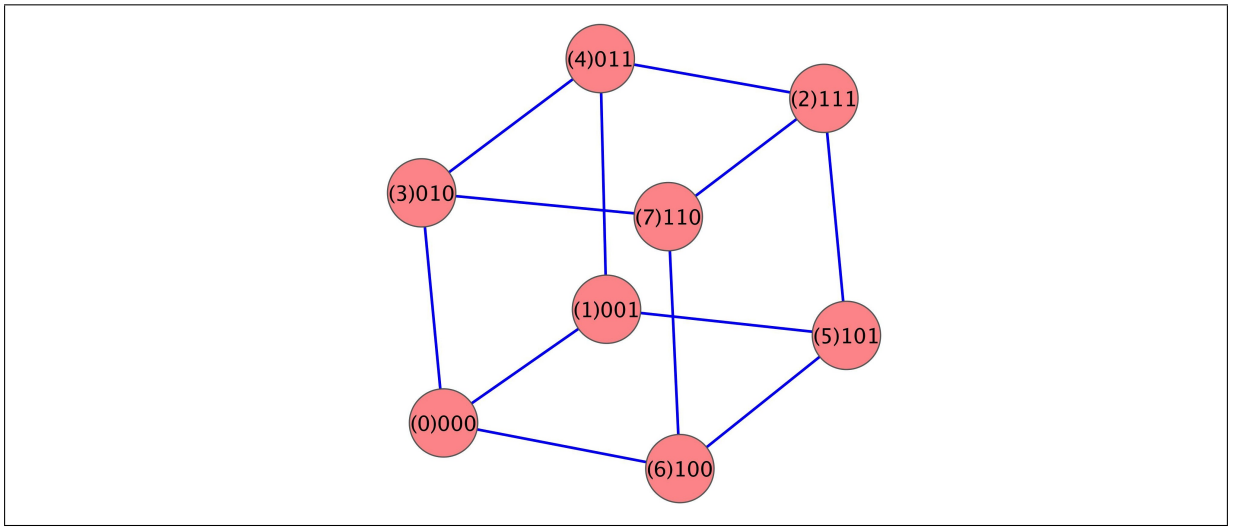

FIGURE B.1: HyperD: 8 Nodes No Network Changes No Hop Restriction

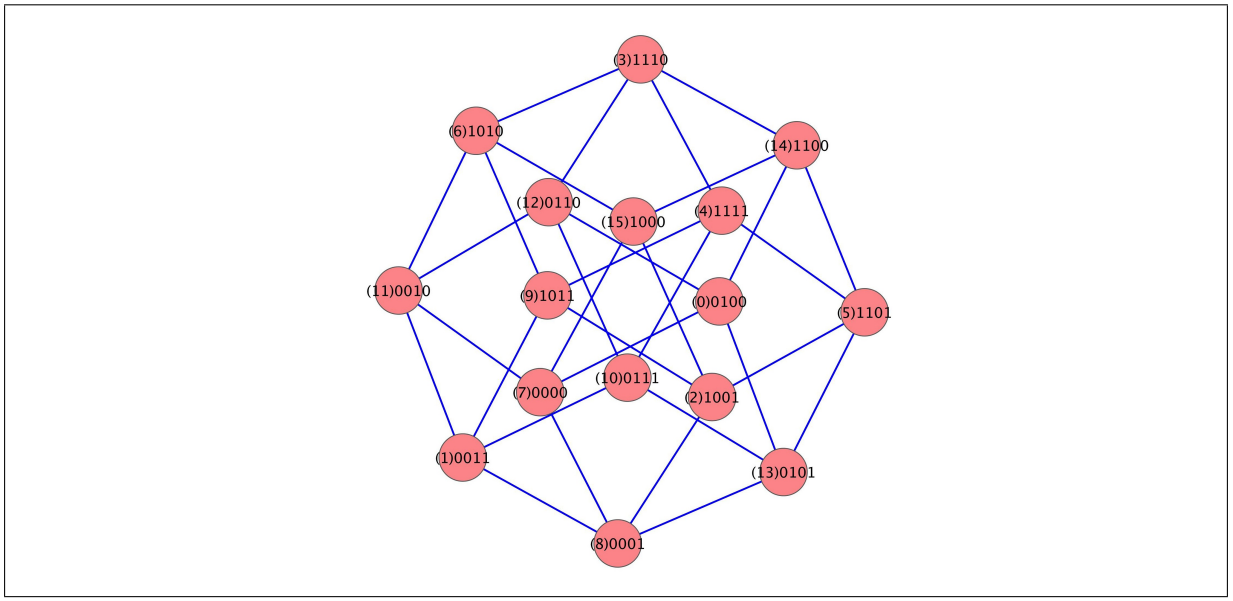

FIGURE B.2: HyperD: 16 Nodes No Network Changes No Hop Restriction

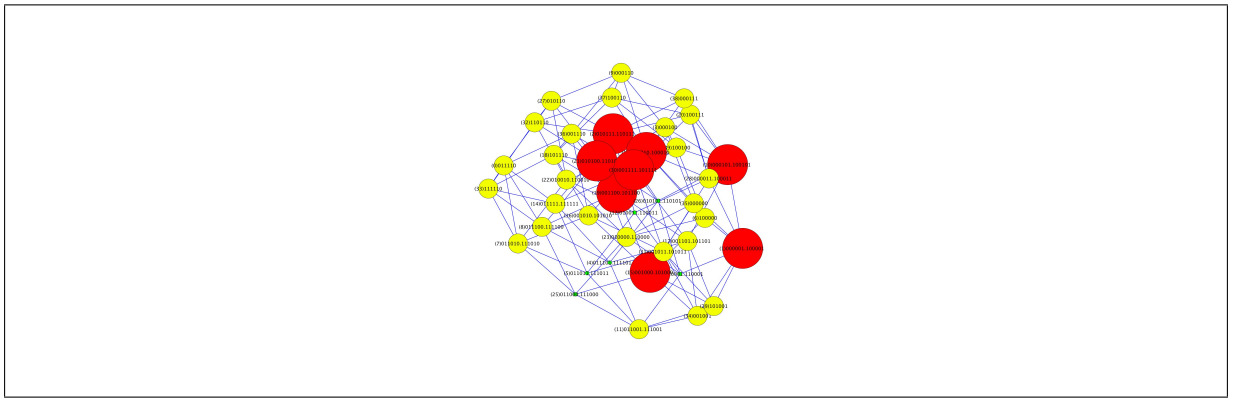

FIGURE B.3: HyperD: 40 Nodes Before Network Changes

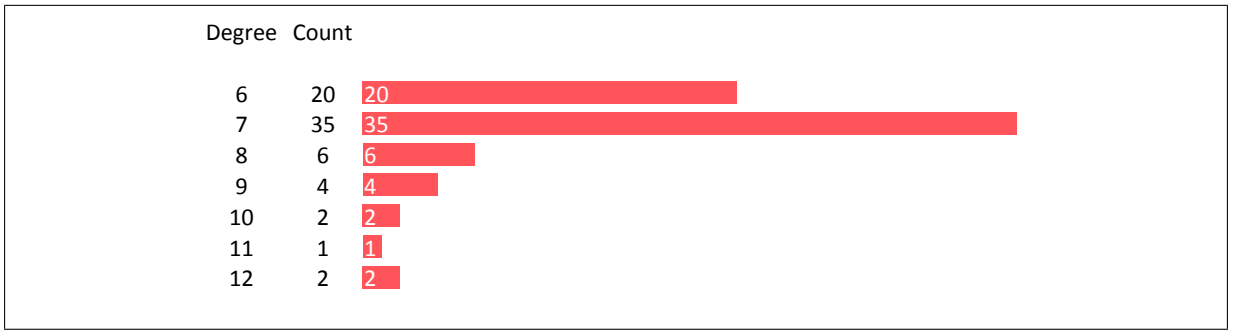

FIGURE B.4: HyperD: 70 Node Degree Distribution

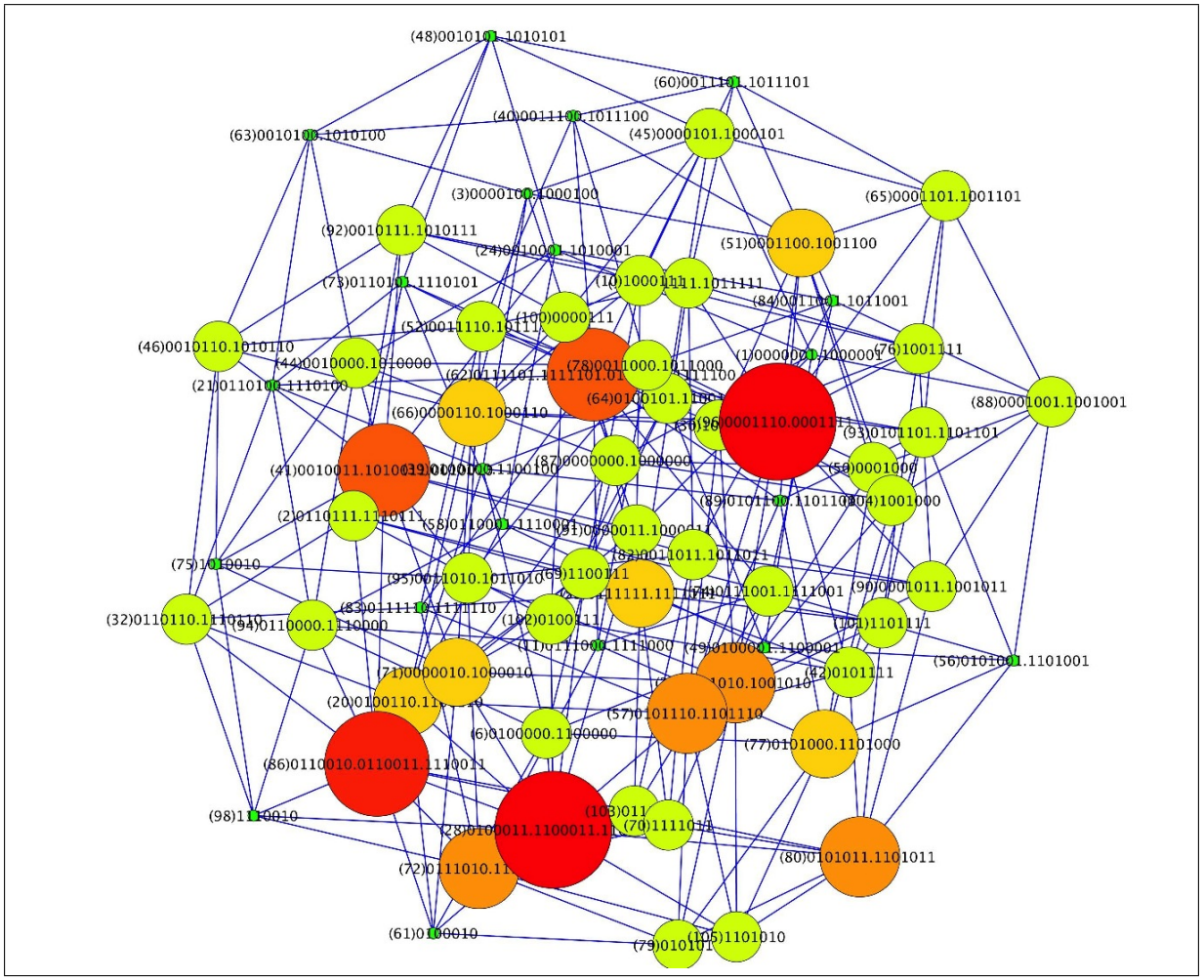

FIGURE B.5: HyperD: 70 Nodes, Start 40 Nodes No Hop Restriction

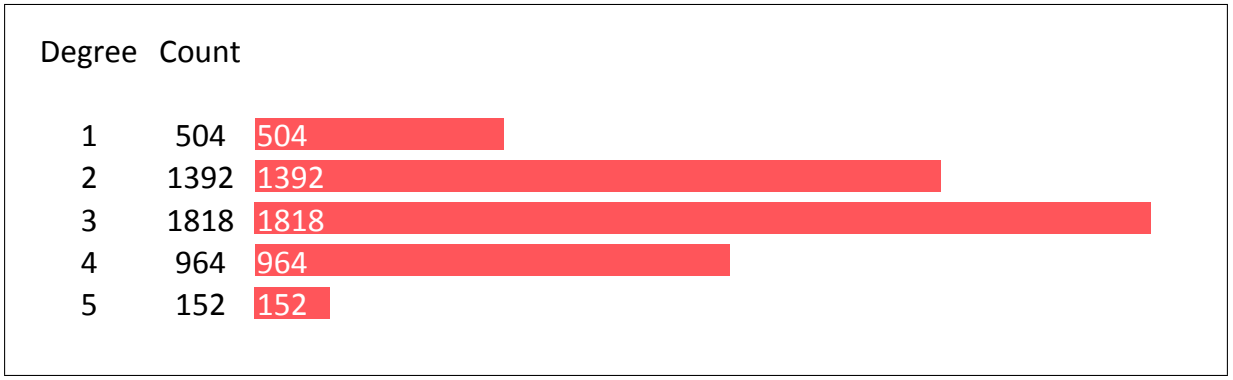

FIGURE B.6: HyperD: 70 Node Shortest Path Length Distribution

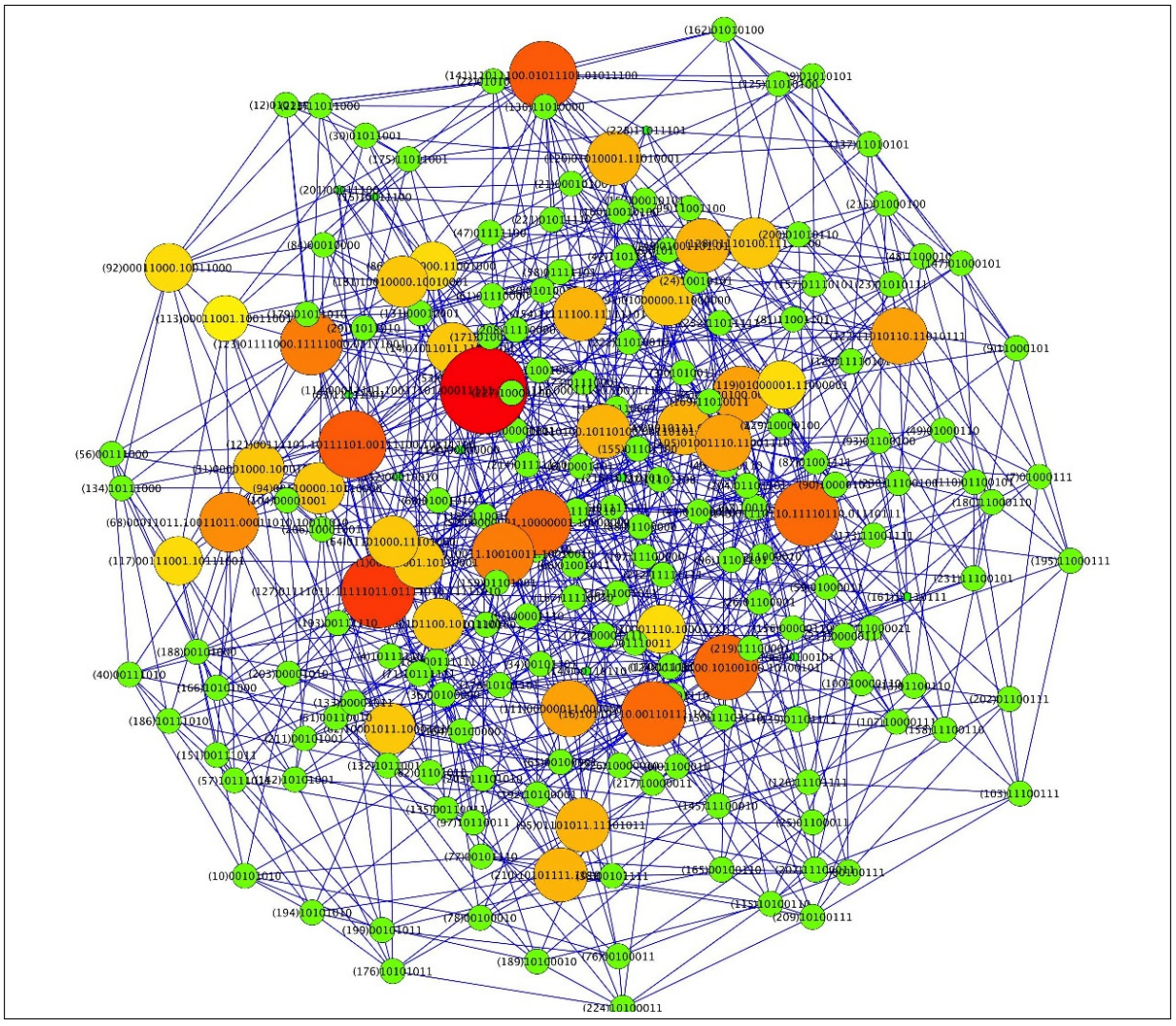

FIGURE B.7: HyperD: 200 Final Nodes, 120 Start Nodes, No Hop Restriction

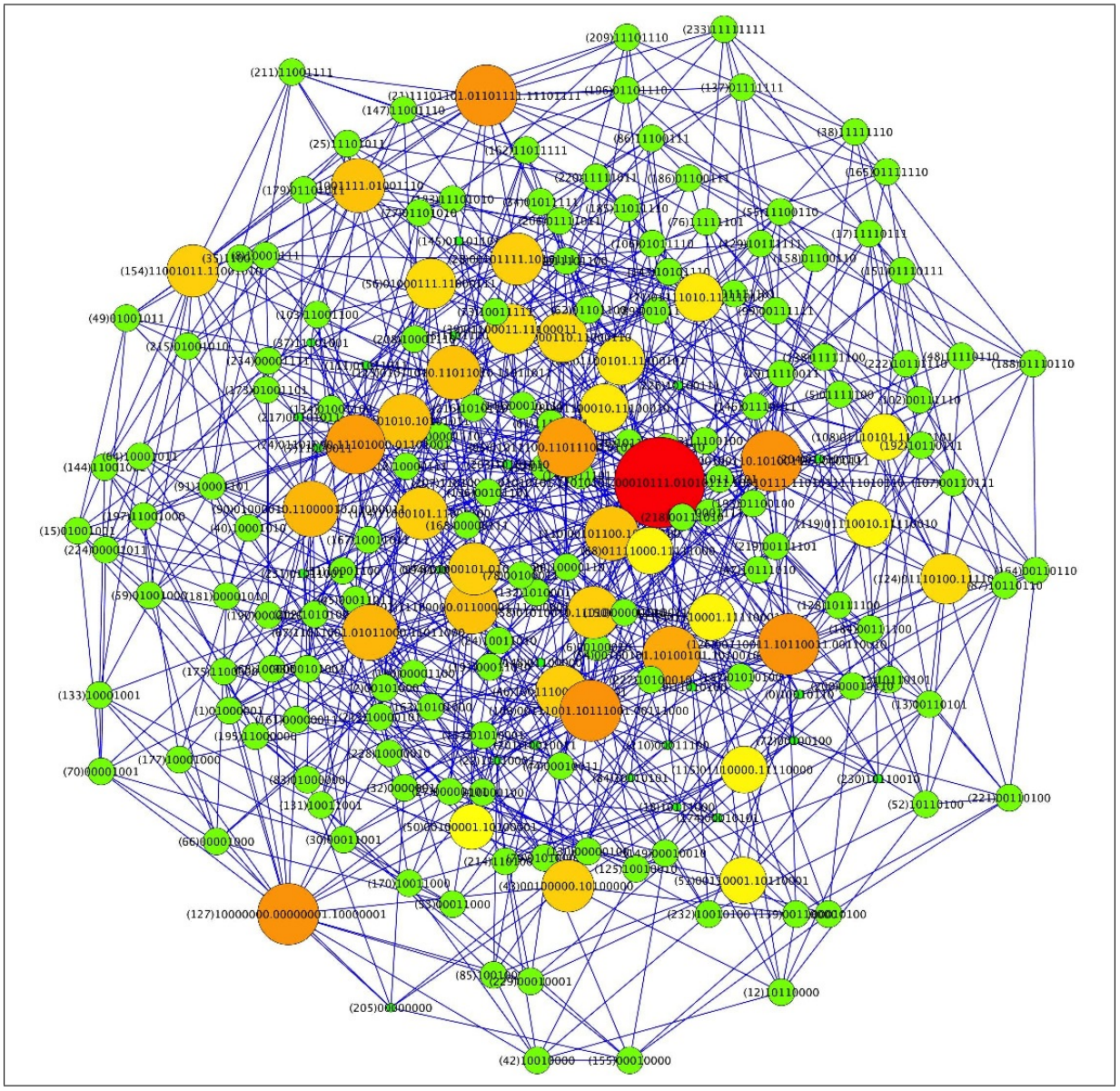

FIGURE B.8: HyperD: 200 Final Nodes, 120 Start Nodes, Two Hop Restriction

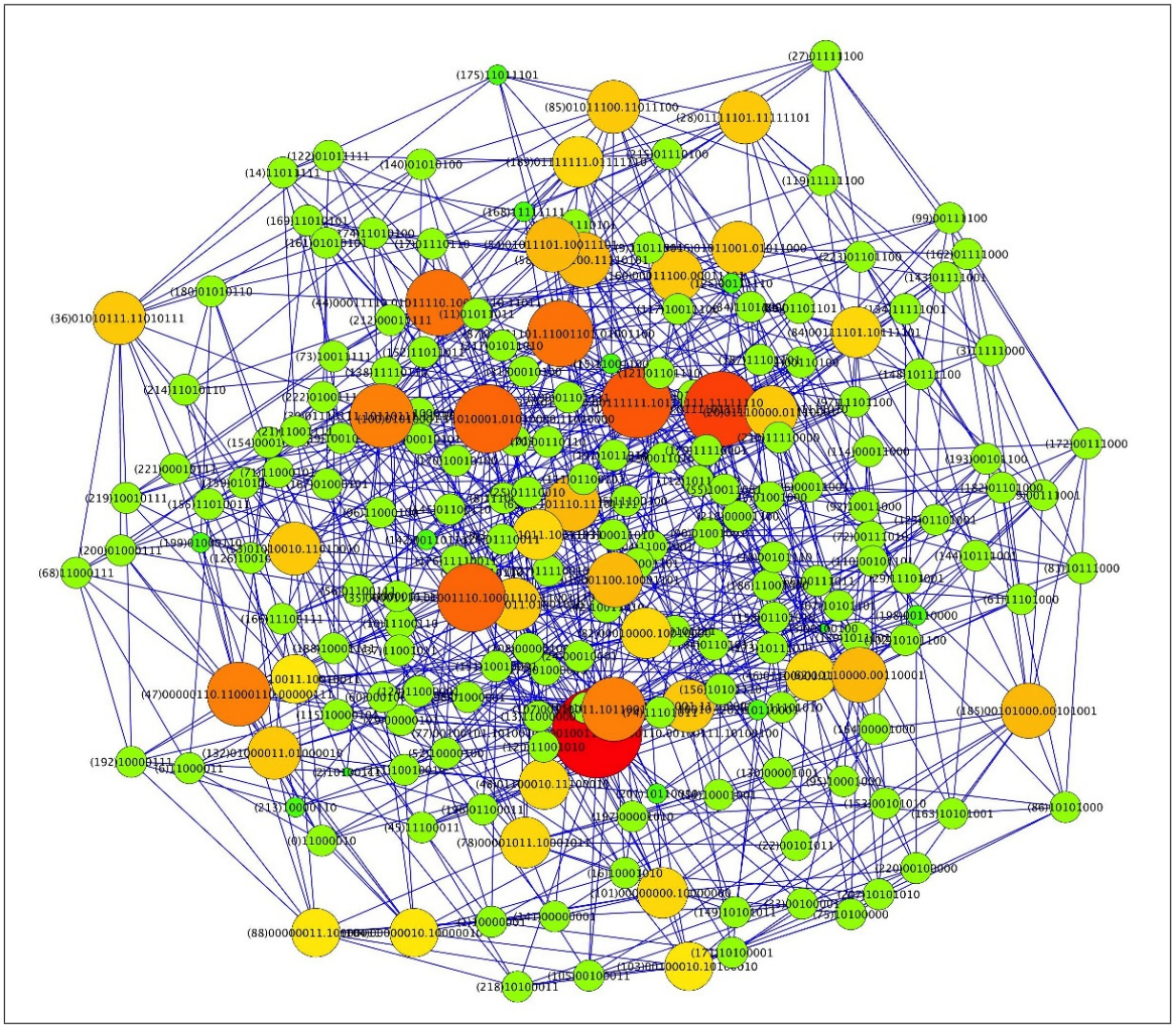

FIGURE B.9: HyperD: 200 Nodes 120 Start One Hop Restriction

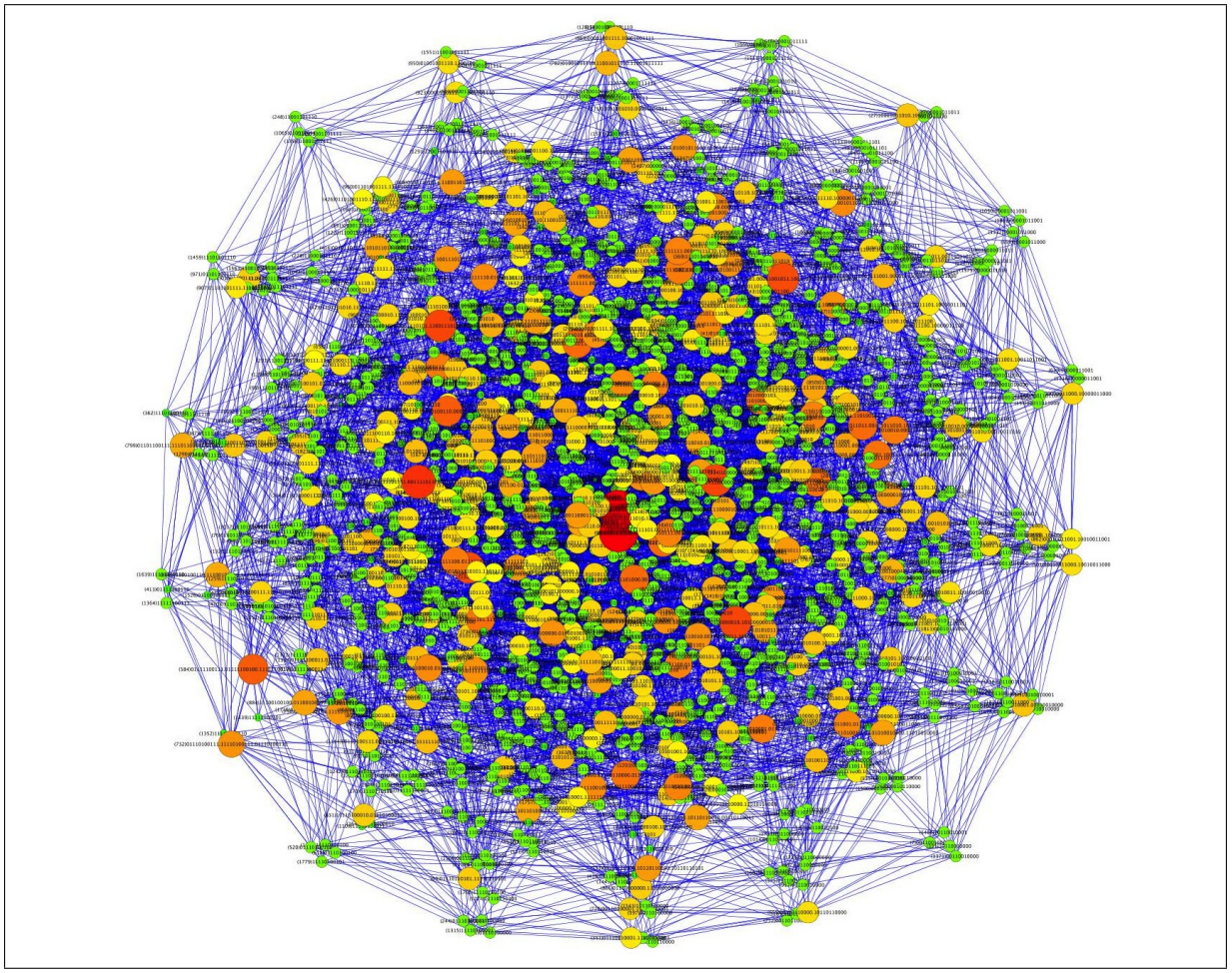

FIGURE B.10: HyperD: 1500 Nodes, 1000 Start Nodes, No Hop Restriction used for Broadcast Illustration

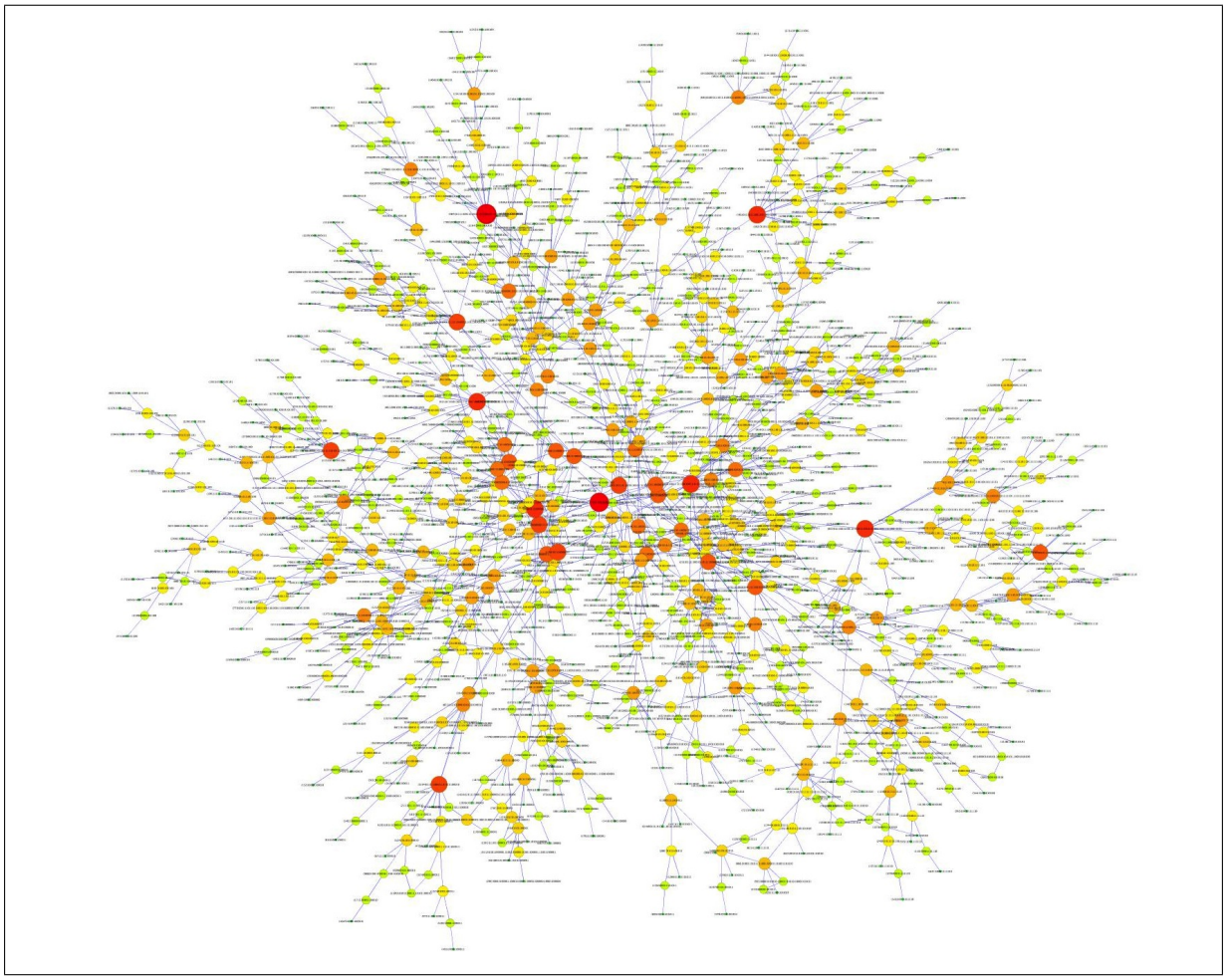

FIGURE B.11: Broadcast Tree Example for the HyperD Network in Figure B.10, 1500 Nodes 1710 Edges

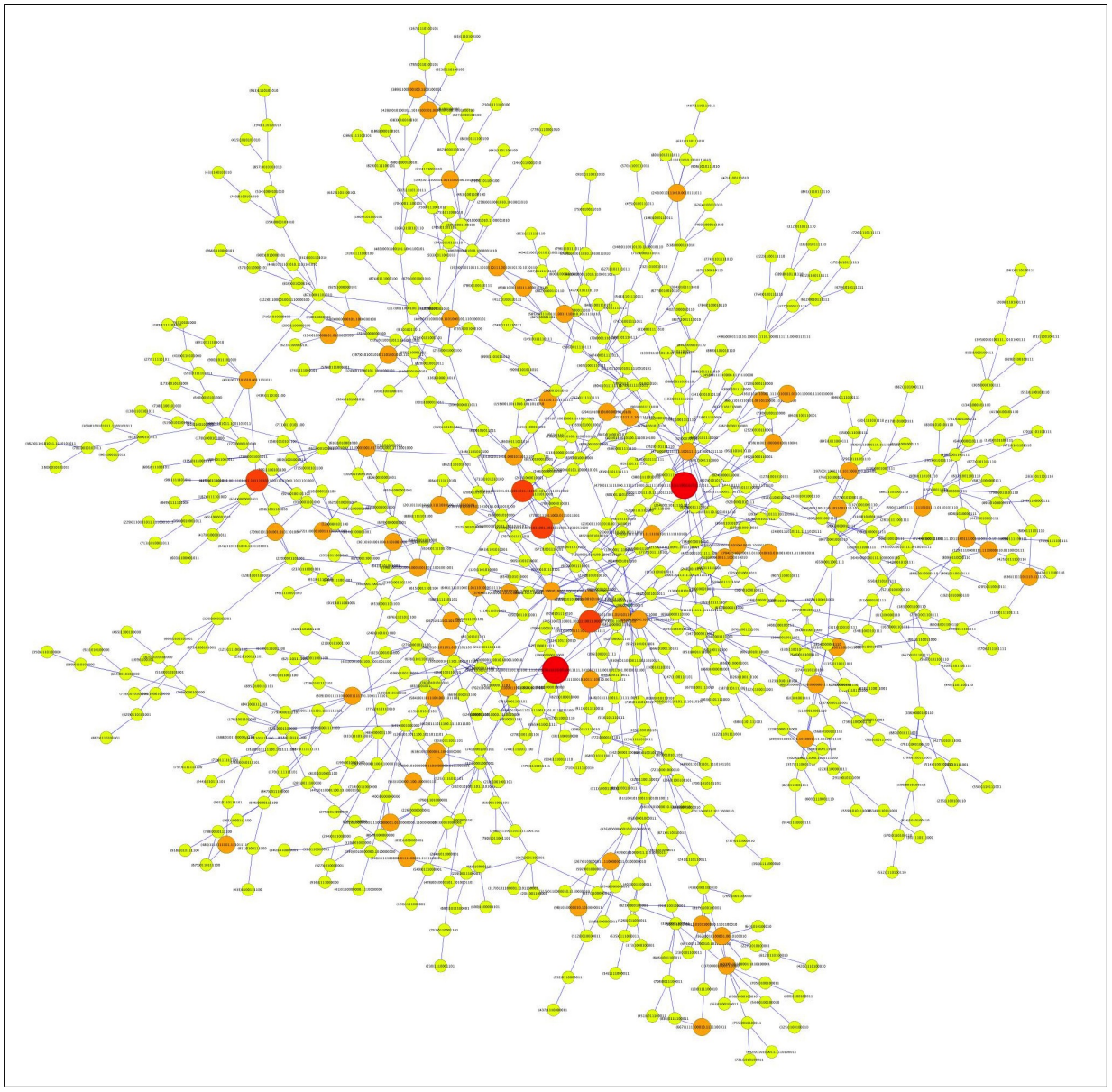

FIGURE B.12: Broadcast Tree Example for a HyperD Network of 800 Nodes. Node Size and Color is Determined by In-degree.

Start Nodes	Total Nodes	Total Paths	Diameter	Vertex Degree	Average Path HyperD	Average Path Optimal	Percentage Difference
10	20	380	4	5	2.132	2.132	0.00%
10	20	380	3	5.3	1.966	1.963	0.15%
10	20	380	4	5.1	2.021	2	1.05%
20	40	1560	4	6.65	2.365	2.34	1.07%
20	40	1560	4	6.85	2.254	2.245	0.40%
20	40	1560	4	6.65	2.304	2.296	0.35%
40	80	6320	5	8.4	2.562	2.521	1.63%
40	80	6320	5	8.05	2.689	2.649	1.51%
40	80	6320	5	8	2.683	2.652	1.17%
80	160	25440	5	9.3	3.059	2.993	2.21%
80	160	25440	6	9	3.196	3.123	2.34%
80	160	25440	6	9.3	3.04	2.975	2.18%
160	320	102080	7	10.4	3.649	3.576	2.04%
160	320	102080	7	10.3	3.642	3.564	2.19%
160	320	102080	7	10.5	3.584	3.531	1.50%
320	640	408960	7	11.9	4.038	3.952	2.18%
320	640	408960	7	11.9	3.958	3.838	3.13%
320	640	408960	7	12	3.949	3.84	2.84%
640	1280	1637120	8	13.1	4.409	4.264	3.40%
640	1280	1637120	8	12.8	4.53	4.395	3.07%
640	1280	1637120	8	13.1	4.437	4.406	0.70%

FIGURE B.13: Average Path Comparison Table for HyperD Networks of Different Sizes with No Hop Restriction on Enter Broadcasts. The Resulting Graph is Shown in Figure 3.13

Start Nodes	Total Nodes	Total Paths	Diameter	Vertex Degree	Average Path HyperD	Average Path Optimal	Percentage Difference
20	40	1560	4	6.8	2.163	2.145	0.84%
20	40	1560	4	6.7	2.221	2.201	0.91%
20	40	1560	4	6.6	2.313	2.299	0.61%
40	80	6320	4	8.2	2.534	2.484	2.01%
40	80	6320	4	8.3	2.343	2.322	0.90%
40	80	6320	4	8.5	2.446	2.406	1.66%
80	160	25440	5	9.6	2.827	2.748	2.87%
80	160	25440	5	9.6	2.817	2.739	2.85%
80	160	25440	5	9.5	2.643	2.598	1.73%
160	320	102080	5	11	3.124	2.992	4.41%
160	320	102080	5	11.3	3.041	2.937	3.54%
160	320	102080	6	11.1	3.288	3.177	3.49%
320	640	408960	6	12.5	3.52	3.36	4.76%
320	640	408960	6	12.5	3.408	3.253	4.76%
320	640	408960	5	14	2.999	2.895	3.59%
640	1280	1637120	7	13.8	3.852	3.635	5.97%
640	1280	1637120	6	14	3.787	3.578	5.84%
640	1280	1637120	6	13.9	3.775	3.57	5.74%

FIGURE B.14: Average Path Comparison Table for HyperD Networks of Different Sizes with One Hop Restriction on Enter Broadcasts. The Resulting Graph is Shown in Figure 3.14

Appendix C

BLWNet Example

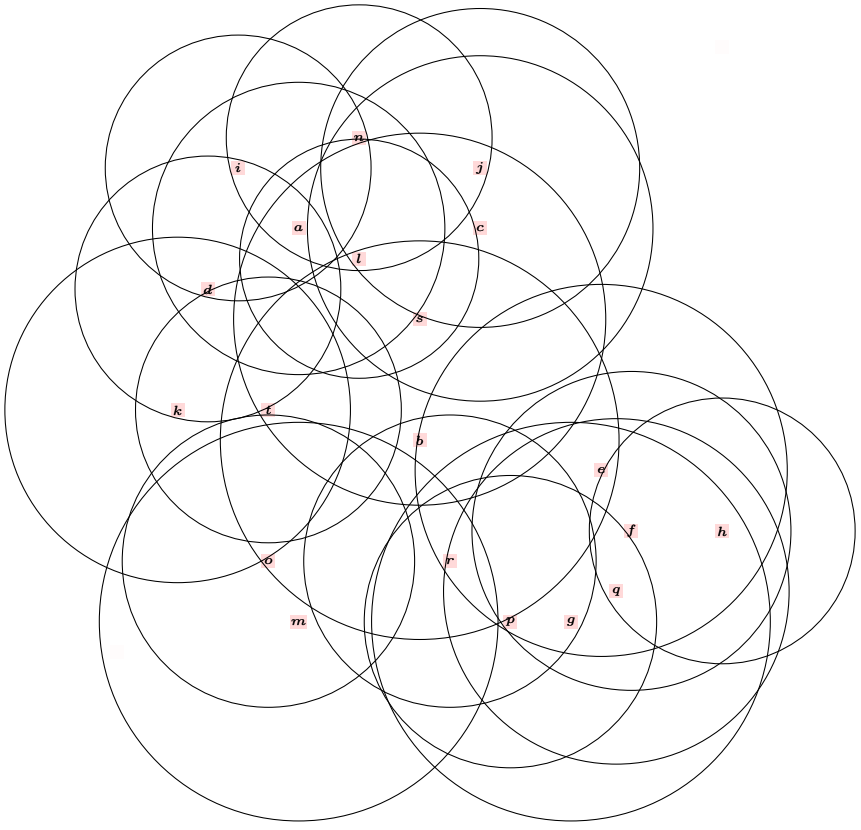

FIGURE C.1: Example of Wireless Ad-Hoc Network. The Position of Each Node Indicates their Geographic Location in Two-Dimensional Space. Each Circle indicates the Range of the Node at its Center.

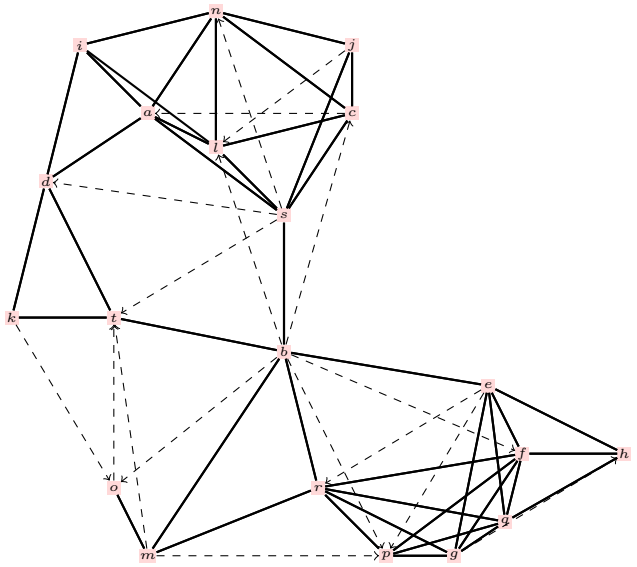

FIGURE C.2: A Graph Structure Modelling the Wireless Network Shown in Figure C.1. Solid Lines Show Bi-Directional Communication and Dashed Lines Uni-Directional Communication.

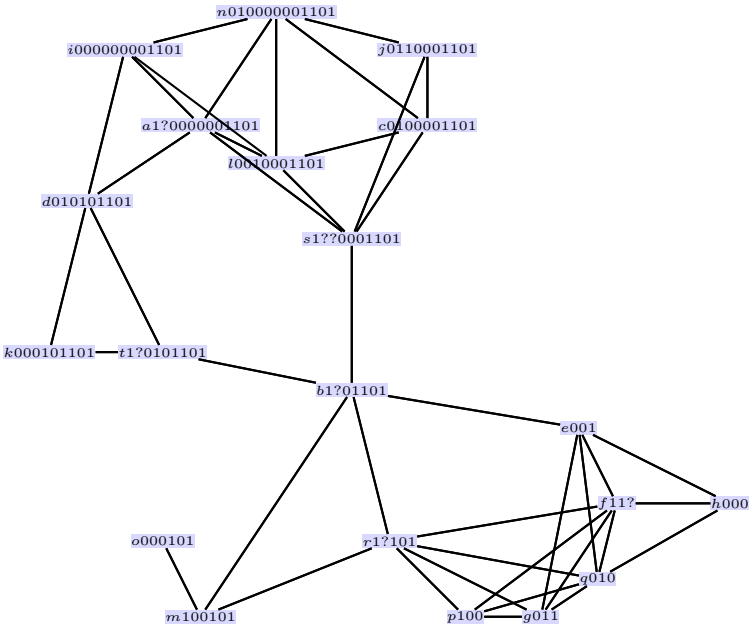

FIGURE C.3: BLWNet Labeled Graph Modelling the Network in Figure C.1.

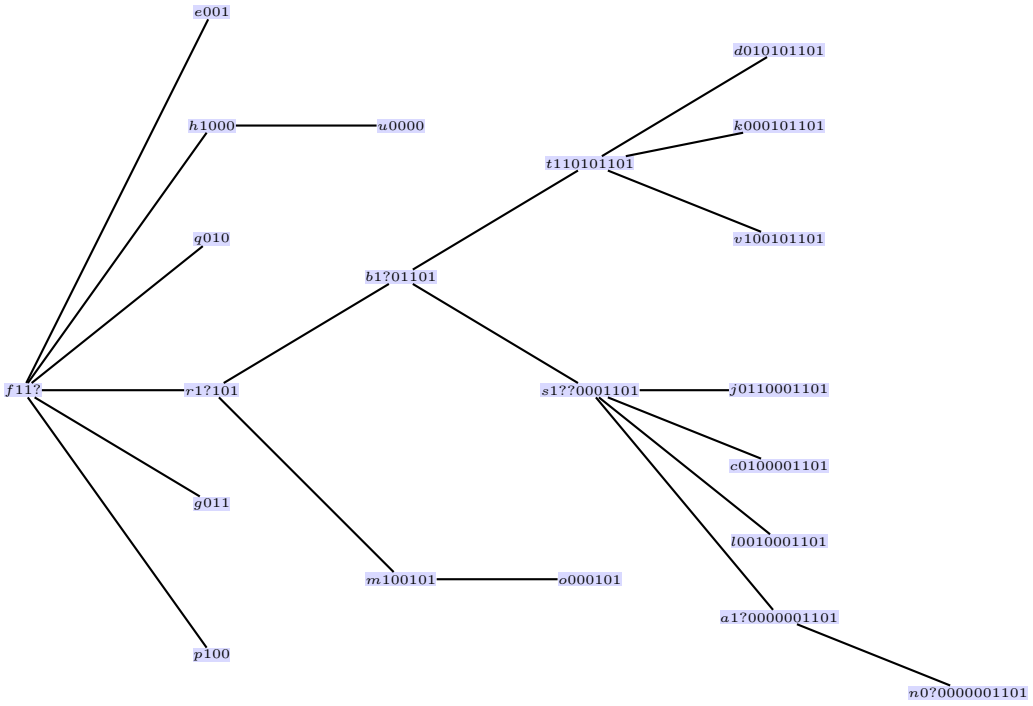

FIGURE C.4: Ancestry Tree for the Labeled Network Shown in Figure C.3.

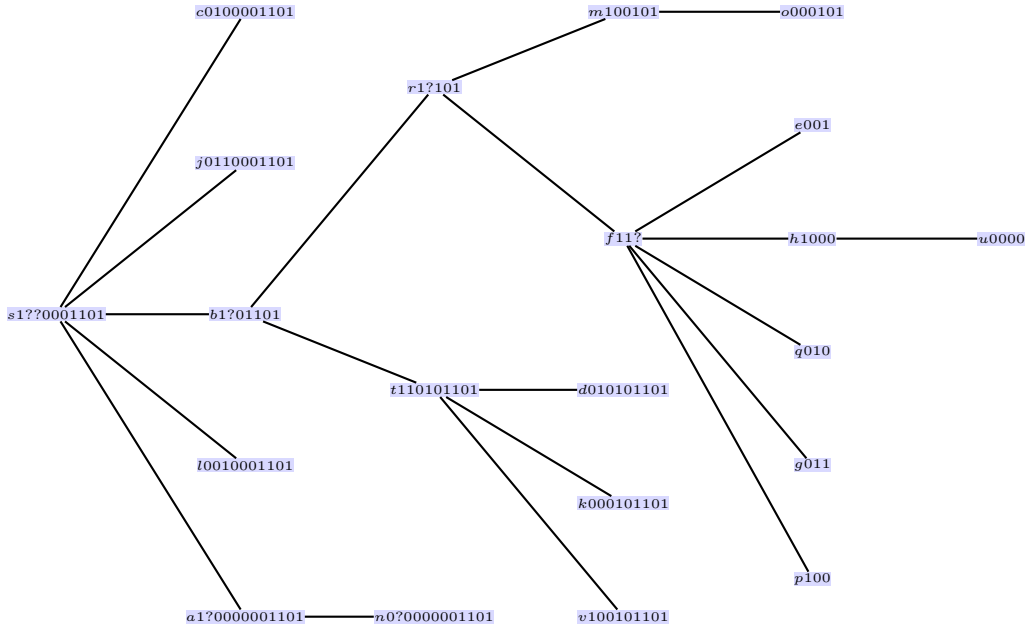

FIGURE C.5: Broadcast Tree Example Using the Ancestry Tree in Figure C.4.

Appendix D

BLWNet Experimental Results

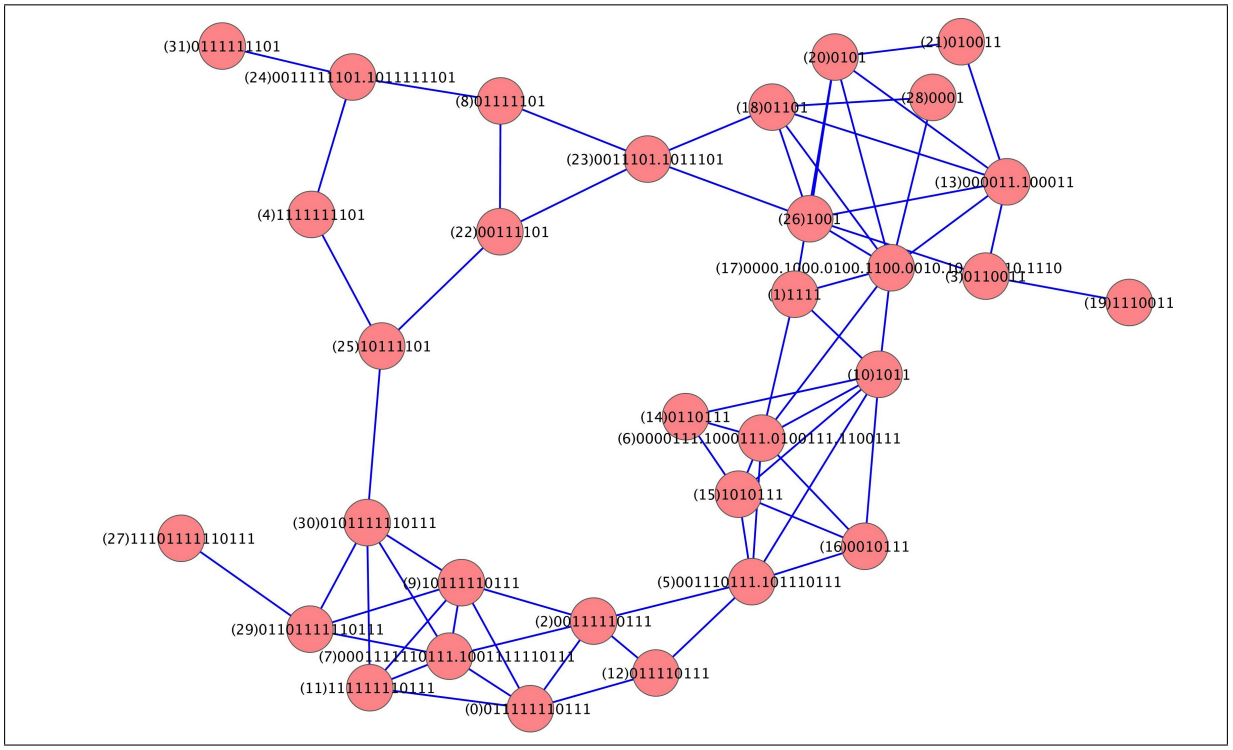

FIGURE D.1: BLWNet: 32 Nodes, Area 25x25, Range 4-8

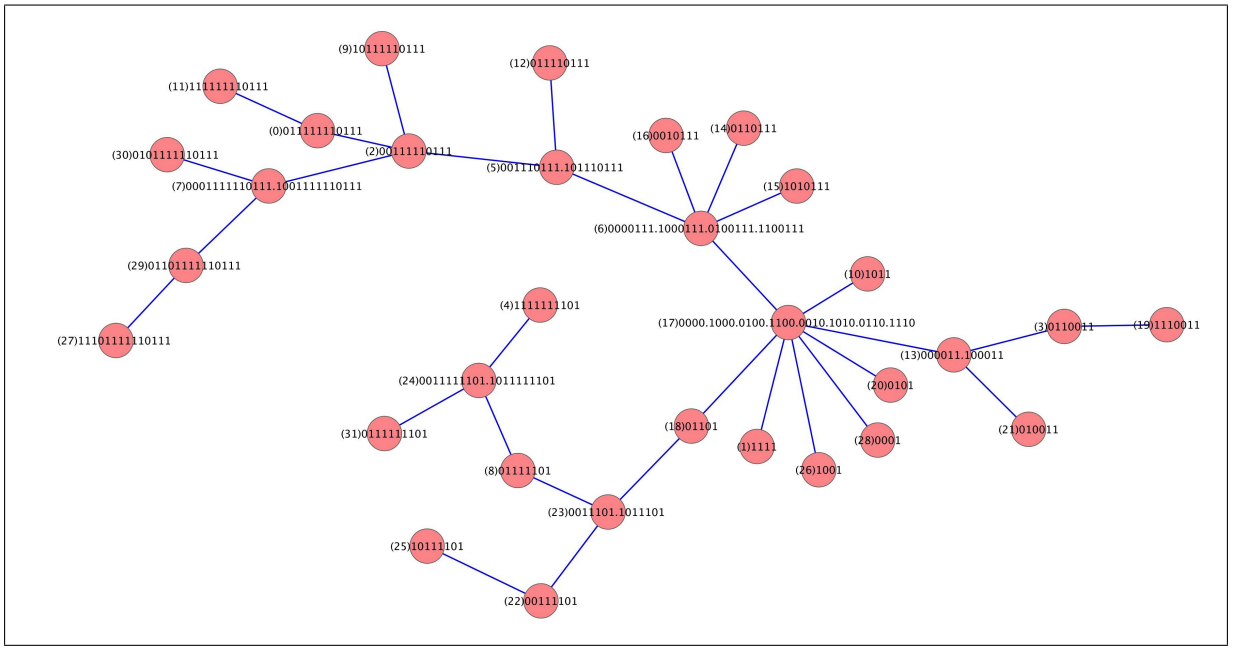

FIGURE D.2: Ancestry Tree for the network

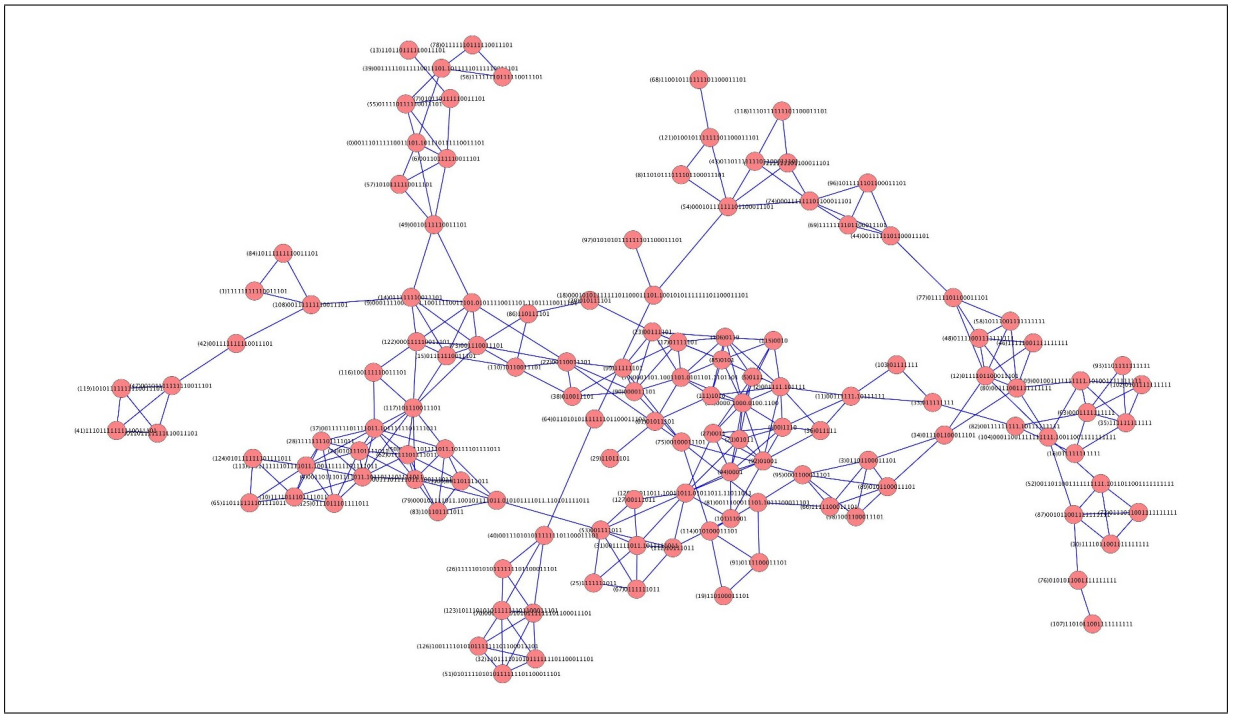

FIGURE D.3: BLWNet: 128 Nodes, Area 40x40, Range 4-8

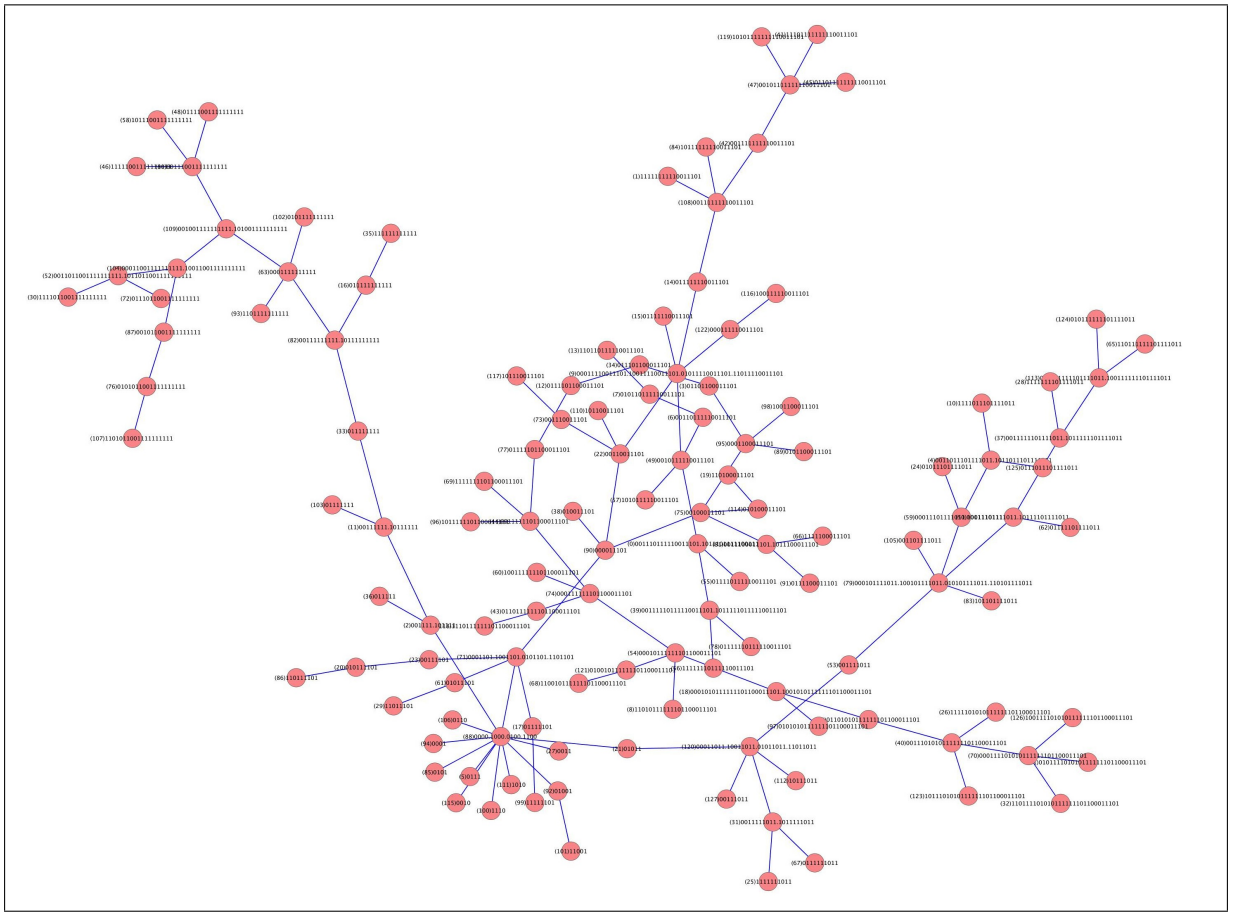

FIGURE D.4: Ancestry Tree for the network

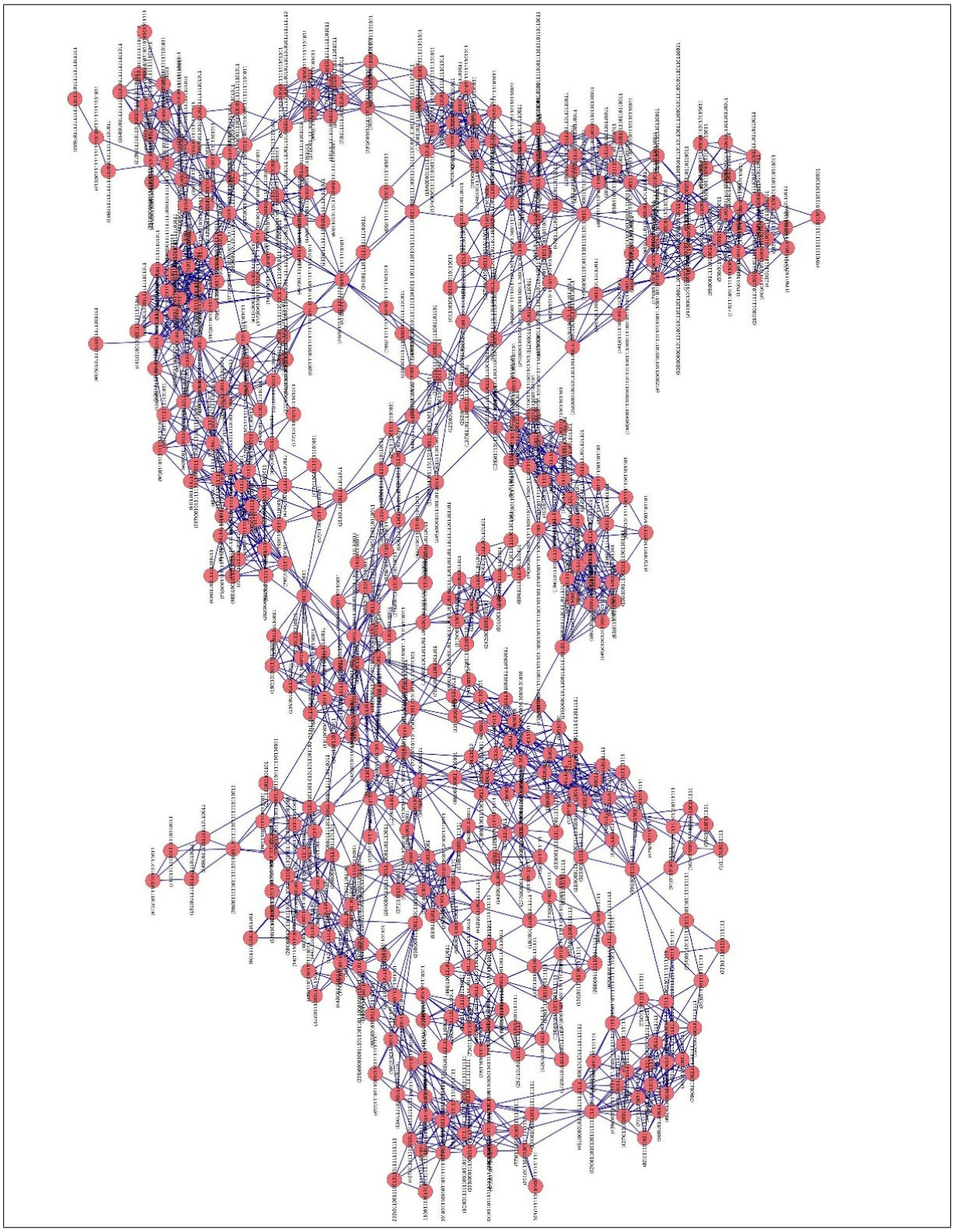

FIGURE D.5: BLWNet: 512 Nodes, Area 60x60, Range 4-8

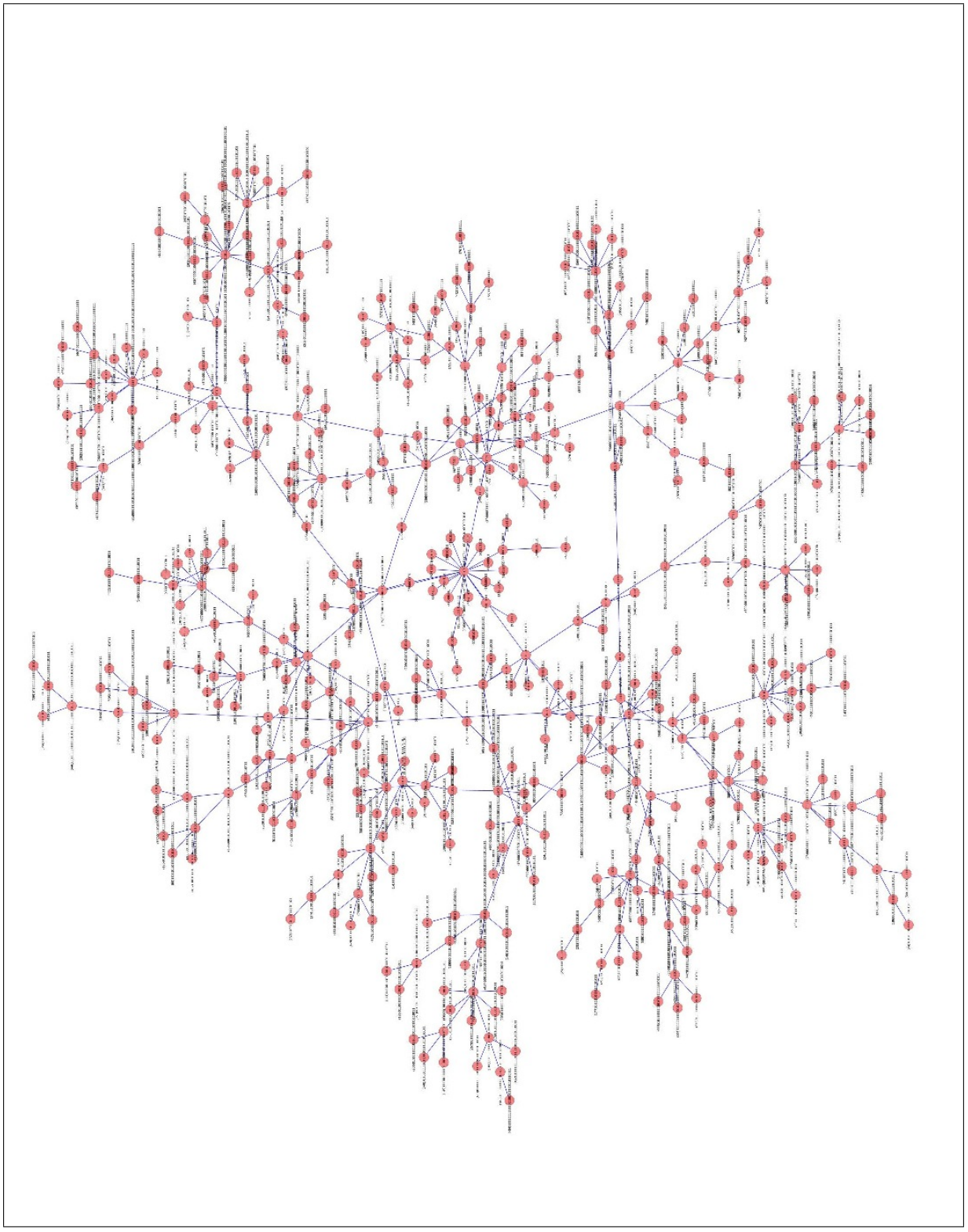

FIGURE D.6: Ancestry Tree for the network

Start Nodes	Total Nodes	Area	Total Paths	Diameter	Vertex Degree	Average Path BLWNET	Average Path Optimal	Percentage Difference
16	16	15	240	5	3.125	3.113	2.733	13.90%
16	16	15	240	5	3.375	2.875	2.867	0.28%
16	16	15	240	6	3.875	2.846	2.642	7.72%
32	32	20	992	10	4.625	3.911	3.641	7.42%
32	32	20	992	8	3.938	3.992	3.583	11.42%
32	32	18	992	5	5.938	3.278	2.692	21.77%
64	64	25	4032	10	5.626	4.717	4.025	17.19%
64	64	25	4032	9	5.781	4.668	3.822	22.14%
64	64	25	4032	10	6.281	4.493	4.151	8.24%
128	128	30	16256	13	8.063	6.421	5.093	26.08%
128	128	30	16256	12	9.469	5.546	4.423	25.39%
128	128	30	16256	10	8.812	5.918	4.598	28.71%
256	256	40	65280	14	10.594	6.571	5.158	27.39%
256	256	40	65280	13	10.641	7.099	5.693	24.70%
256	256	40	65280	15	10.961	6.822	5.233	30.36%
512	512	50	261632	14	14.551	7.561	5.957	26.93%
512	512	55	261632	19	11.453	8.689	6.955	24.93%
512	512	55	261632	17	11.738	8.805	7.042	25.04%
1024	1024	70	1047552	20	14.561	10.621	8.111	30.95%
1024	1024	70	1047552	21	14.912	11.099	8.147	36.23%
1024	1024	70	1047552	20	14.359	11.789	8.163	44.42%

FIGURE D.7: Average Path Comparison Table with No Topology Changes

Start Nodes	Total Nodes	Area	Total Paths	Diameter	Vertex Degree	Average Path BLWNET	Average Path Optimal	Percentage Difference
10	16	15	240	4	5	2.542	2.042	24.49%
10	16	15	240	6	4	2.554	2.392	6.77%
10	16	15	240	4	4.5	2.201	2.058	6.95%
20	32	18	992	6	4.938	3.438	2.766	24.30%
20	32	18	992	5	5.875	5.186	2.579	101.09%
20	32	18	992	8	5.5	4.121	3.107	32.64%
50	64	25	4032	7	7.031	8.296	3.547	133.89%
50	64	25	4032	10	6.094	5.145	4.281	20.18%
50	64	25	4032	10	6.406	8.375	4.526	85.04%
100	128	30	16256	10	8.875	9.374	4.432	111.51%
100	128	30	16256	12	9.39	7.051	4.536	55.45%
100	128	30	16256	9	8.36	8.546	4.409	93.83%
210	256	40	65280	13	10.602	8.491	5.532	53.49%
210	256	40	65280	13	10.57	13.578	5.165	162.88%
210	256	40	65280	13	9.992	10.793	5.567	93.87%
440	512	50	261632	14	14.203	8.219	5.769	42.47%
440	512	50	261632	15	13.738	14.912	6.453	131.09%
440	512	50	261632	16	14.082	9.659	6.131	57.54%
900	1024	70	1047552	20	14.732	12.667	8.043	57.49%
900	1024	70	1047552	20	14.303	18.17	8.192	121.80%
900	1024	70	1047552	20	15.076	13.183	8.163	61.50%

FIGURE D.8: Average Path Comparison Table with Topology Changes

Start Nodes	Total Nodes	Area	Total Paths	Diameter	Vertex Degree	Average Path BLWNET	Average Path Optimal	Percentage Difference
10	16	15	240	6	3.875	2.825	2.567	10.05%
10	16	15	240	6	4.25	4.058	2.35	72.68%
10	16	15	240	4	4.875	2.179	2.017	8.03%
20	32	18	992	5	8.186	2.686	2.212	21.43%
20	32	18	992	6	6.938	3.115	2.541	22.59%
20	32	18	992	6	6.187	5.568	2.702	106.07%
50	64	25	4032	9	6.594	4.548	3.572	27.32%
50	64	25	4032	13	6.343	5.085	4.694	8.33%
50	64	25	4032	9	6.875	6.454	3.727	73.17%
100	128	30	16256	11	7.953	9.837	4.864	102.24%
100	128	30	16256	10	8.312	6.694	4.383	52.73%
100	128	30	16256	10	8.501	8.371	4.242	97.34%
210	256	40	65280	13	10.523	11.831	5.624	110.37%
210	256	40	65280	12	10.328	8.099	5.393	50.18%
210	256	40	65280	14	9.828	11.345	5.438	108.62%
440	512	50	261632	15	13.703	10.381	6.192	67.65%
440	512	50	261632	15	13.5	8.633	6.251	38.11%
440	512	50	261632	16	13.715	12.114	6.408	89.04%
900	1024	70	1047552	21	14.924	14.457	8.226	75.75%
900	1024	70	1047552	20	14.637	13.029	8.313	56.73%
900	1024	70	1047552	21	14.596	12.069	8.235	46.56%

FIGURE D.9: Average Path Comparison Table with Topology Changes and Partial Relabeling

Bibliography

[1] John Bufford and Heather Yu. Peer-to-peer networking: Synopsis and research directions. In Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 3–46. Springer Publishing Company, Incorporated, 2009.

[2] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[3] Ken Y. K. Hui, John C. S. Lui, and David K. Y. Yau. Small-world overlay p2p networks: construction, management and handling of dynamic flash crowds. *Comput. Netw.*, 50 (15):2727–2746, October 2006. ISSN 1389-1286. doi: 10.1016/j.comnet.2005.10.010. URL http://dx.doi.org/10.1016/j.comnet.2005.10.010.

[4] Prasanna Ganesan, Qixiang Sun, and Hector Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *INFOCOM*, 2003. doi: http://www.ieee-infocom.org/2003/papers/31_01.PDF.

[5] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, January 2002. ISSN 1089-7801.

[6] Jian Liang, Rakesh Kumar, and Keith W. Ross. The fasttrack overlay: a measurement study. *Comput. Netw.*, 50(6):842–858, 2006. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2005.07.014. URL http://portal.acm.org/citation.cfm?id=1141110.

[7] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM. ISBN 1-58113-735-4. doi: 10.1145/863955.864000. URL http://doi.acm.org/10.1145/863955.864000.

[8] Ruggero Morselli, Bobby Bhattacharjee, Aravind Srinivasan, and Michael A. Marsh. Efficient lookup on unstructured topologies. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, pages 77–86, New York, NY, USA, 2005. ACM. ISBN 1-58113-994-2. doi: 10.1145/1073814.1073828. URL http://doi.acm.org/10.1145/1073814.1073828.

[9] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *INTERNATIONAL WORKSHOP ON DESIGNING PRIVACY ENHANCING TECHNOLOGIES:DESIGN*

ISSUES IN ANONYMITY AND UNOBSERVABILITY, pages 46–66. Springer-Verlag New York, Inc., 2001.

[10] J. A. Pouwelse, P. Garbacki, J. Yang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. Van Steen, and H. J. Sips. Tribler: A social-based peer-to-peer system. In *In The 5th International Workshop on Peer-to-Peer Systems (IPTPS06*, 2006.

[11] Alexander Loser, Steffen Staab, and Christoph Tempich. Semantic social overlay networks. *IEEE J.Sel. A. Commun.*, 25(1):5–14, January 2007. ISSN 0733-8716. doi: 10.1109/JSAC.2007.070102. URL http://dx.doi.org/10.1109/JSAC.2007.070102.

[12] 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.

[13] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, SPAA '97, pages 311–320, New York, NY, USA, 1997. ACM. ISBN 0-89791-890-8. doi: 10.1145/258492.258523. URL http://doi.acm.org/10.1145/258492.258523.

[14] Xiaozhou Li, Jayadev Misra, and C. Greg Plaxton. Active and concurrent topology maintenance. In *IN PROC. 18TH ANN. CONFERENCE ON DISTRIBUTED COMPUTING (DISC*, pages 320–334. Springer, 2004.

[15] B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE J.Sel. A. Commun.*, 22(1):41–53, September 2006. ISSN 0733-8716. doi: 10.1109/JSAC.2003.818784. URL http://dx.doi.org/10.1109/JSAC.2003.818784.

[16] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42800-3. URL http://dl.acm.org/citation.cfm?id=646591.697650.

[17] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, September 2003. ISSN 0163-5808. doi: 10.1145/945721.945729. URL http://doi.acm.org/10.1145/945721.945729.

[18] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, August 2005. ISSN 0146-4833. doi: 10.1145/1090191.1080102. URL http://doi.acm.org/10.1145/1090191.1080102.

[19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001. ISSN 0146-4833. doi: 10.1145/964723.383071. URL http://doi.acm.org/10.1145/964723.383071.

[20] Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. Dks (n, k, f): A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In *Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, CCGRID '03, pages 344–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1919-9. URL http://dl.acm.org/citation.cfm?id=791231.792434.

[21] Bruno Carton and Valentin Mesaros. Improving the scalability of logarithmic-degree dht-based peer-to-peer networks. In *In Proc. of EUROPAR*, pages 1060–1067, 2004.

[22] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251460.1251470.

[23] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4. URL http://dl.acm.org/citation.cfm?id=646334.687801.

[24] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of dht routing tables. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 99–114, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251203.1251211.

[25] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. One hop lookups for peer-to-peer overlays. In *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*, HOTOS'03, pages 2–2, Berkeley, CA, USA, 2003. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251054.1251056.

[26] Ben Leong, Barbara Liskov, and Erik D. Demaine. Epichord: Parallelizing the chord lookup algorithm with reactive routing state management. *Comput. Commun.*, 29(9): 1243–1259, May 2006. ISSN 0140-3664. doi: 10.1016/j.comcom.2005.10.002. URL http://dx.doi.org/10.1016/j.comcom.2005.10.002.

[27] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[28] Ittai Abraham, Ankur Badola, Danny Bickson, Dahlia Malkhi, Sharad Maloo, and Saar Ron. Practical locality-awareness for large scale information sharing. In *Proceedings of the 4th international conference on Peer-to-Peer Systems*, IPTPS'05, pages 173–181, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29068-0, 978-3-540-29068-1. doi: 10.1007/11558989_16. URL http://dx.doi.org/10.1007/11558989_16.

[29] Luiz R. Monnerat and Claudio L. Amorim. D1ht: a distributed one hop hash table. In *Proceedings of the 20th international conference on Parallel and distributed processing*,

IPDPS'06, pages 40–40, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 1-4244-0054-6. URL http://dl.acm.org/citation.cfm?id=1898953.1898974.

[30] Luis Garcs-Erice, Keith W. Ross, Ernst W. Biersack, Pascal Felber, and Guillaume Urvoy-Keller. Topology-centric look-up service. In Burkhard Stiller, Georg Carle, Martin Karsten, and Peter Reichl, editors, *Networked Group Communication*, volume 2816 of *Lecture Notes in Computer Science*, pages 58–69. Springer, 2003. ISBN 3-540-20051-7. URL http://dblp.uni-trier.de/db/conf/ngc/ngc2003.html#Garces-EriceRBFU03.

[31] Zhiyong Xu, Rui Min, and Yiming Hu. HIERAS: A DHT Based Hierarchical P2P Routing Algorithm. In *32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan*, pages 187–. IEEE Computer Society, 2003.

[32] Marc Sanchez Artigas, Pedro Garcia Lopez, Jordi Pujol Ahullo, and Antonio F. Gomez Skarmeta. Cyclone: A Novel Design Schema for Hierarchical DHTs. In *P2P '05: 5th IEEE International Conference on Peer-to-Peer Computing*, pages 49–56, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2376-5.

[33] Prasanna Ganesan, Krishna Gummadi, and Hector Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04, pages 263–272, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2086-3. URL http://dl.acm.org/citation.cfm?id=977400.978014.

[34] Michael J. Freedman and David Mazieres. Sloppy hashing and self-organizing clusters. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, 20-21 February 2003.

[35] Abhishek Kumar, Shashidhar Merugu, Jun (Jim) Xu, and Xingxing Yu. Ulysses: A robust, low-diameter, low-latency peer-ti-peer network. In *Proceedings of the 11th IEEE International Conference on Network Protocols*, ICNP '03, pages 258–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2024-3. URL http://dl.acm.org/citation.cfm?id=951950.952227.

[36] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, PODC '02, pages 183–192, New York, NY, USA, 2002. ACM. ISBN 1-58113-485-1. doi: http://doi.acm.org/10.1145/571825.571857. URL http://doi.acm.org/10.1145/571825.571857.

[37] M. Frans Kaashoek and David R. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. In *IPTPS*, pages 98–107, 2003.

[38] Anh-Tuan Gai and Laurent Viennot. Broose: A practical distributed hashtable based on the de-bruijn topology. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, P2P '04, pages 167–164, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2156-8. doi: 10.1109/P2P.2004.10. URL http://dx.doi.org/10.1109/P2P.2004.10.

[39] Pierre Fraigniaud and Philippe Gauron. D2b: a de bruijn based content-addressable network. *Theor. Comput. Sci.*, 355(1):65–79, 2006. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2005.12.006.

[40] Moni Naor and Udi Wieder. Novel architectures for p2p applications: The continuous-discrete approach. *ACM Trans. Algorithms*, 3(3), August 2007. ISSN 1549-6325. doi: 10.1145/1273340.1273350. URL http://doi.acm.org/10.1145/1273340.1273350.

[41] Haiying Shen, Cheng-Zhong Xu, and Guihai Chen. Cycloid: a constant-degree and lookup-efficient p2p overlay network. *Perform. Eval.*, 63(3):195–216, 2006. ISSN 0166-5316. doi: http://dx.doi.org/10.1016/j.peva.2005.01.004.

[42] Dongsheng Li, Xicheng Lu, and Jie Wu. Fissione: a scalable constant degree and low congestion dht scheme based on kautz graphs. In *INFOCOM*, pages 1677–1688, 2005.

[43] Deke Guo, Jie Wu, Honghui Chen, and Xueshan Luo. Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree. In *INFOCOM*, pages 821–829, 2007.

[44] Yiming Zhang, Xicheng Lu, and Dongsheng Li. Sky: efficient peer-to-peer networks based on distributed kautz graphs. *Science in China Series F: Information Sciences*, 52(4):588–601, 2009.

[45] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM. ISBN 1-58113-411-8. doi: http://doi.acm.org/10.1145/383059.383072.

[46] Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., California, USA, 1992.

[47] Youcef Saad and Martin H. Schultz. Topological properties of hypercubes. *IEEE Trans. Comput.*, 37(7):867–872, 1988. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/12.2234.

[48] 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.

[49] Abbe Mowshowitz, Akira Kawaguchi, Andi Toce, Andrew Nagel, Paul Stone, Patrick Dantressangle, and Graham Bent. Network topology as a cost factor in query optimization. In *The Fifth Annual Conference of the International Technology Alliance (ACITA'11)*, UMD, College Park, Maryland, USA, USA, September 2011.

[50] Shyue-Ming Tang, Yue-Li Wang, and Yung-Ho Leu. Optimal independent spanning trees on hypercubes. *J. Inf. Sci. Eng.*, 20(1):143–156, 2004.

[51] S. Lennart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Comput.*, 38:1249–1268, September 1989. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/12.29465. URL http://dx.doi.org/10.1109/12.29465.

[52] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *ICPP*, pages 640–648, 1986.

[53] Chien chun Su and Kang G. Shin. Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes. *IEEE Transactions on Computers*, 45:672–683, 1995.

[54] J. Bruck, R. Cypher, and D. Soroker. Running algorithms efficiently on faulty hypercubes. In *SPAA '90: Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, pages 37–44, New York, NY, USA, 1990. ACM. ISBN 0-89791-370-1. doi: http://doi.acm.org/10.1145/97444.97455.

[55] S. Lakshmivarahan and Sudarshan K. Dhall. Ring, torus and hypercube architectures/algorithms for parallel computing. *Parallel Comput.*, 25(13-14):1877–1906, 1999. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(99)00069-1.

[56] 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. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/0743-7315(91)90033-6.

[57] 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.

[58] 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. ISBN 978-0-7695-3404-6. doi: http://dx.doi.org/10.1109/SASO.2008.44.

[59] 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. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2006.04.018.

[60] 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.

[61] 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. ISBN 0-7695-2679-9. doi: 10.1109/P2P.2006.17. URL http://portal.acm.org/citation.cfm?id=1157740.1158232.

[62] J. M. Kumar and L. M. Patnaik. Extended hypercube: A hierarchical interconnection network of hypercubes. *IEEE Trans. Parallel Distrib. Syst.*, 3(1):45–57, 1992. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.113081.

[63] Andi Toce, Abbe Mowshowitz, Paul Stone, Patrick Dantressangle, and Graham Bent. HyperD: a hypercube topology for dynamic distributed federated databases. In *The Fifth Annual Conference of the International Technology Alliance (ACITA'11)*, UMD, College Park, Maryland, USA, USA, September 2011.

[64] Christopher C. Cipriano and Teofilo F. Gonzalez. Multicasting in the hypercube, chord and binomial graphs. *Inf. Process. Lett.*, 110(17):774–777, 2010. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2010.06.009.

[65] Shih-Hsien Sheu and Chang-Biau Yang. Multicast algorithms for hypercube multiprocessors. *J. Parallel Distrib. Comput.*, 61:137–149, January 2001. ISSN 0743-7315. doi: http://dx.doi.org/10.1006/jpdc.2000.1668. URL http://dx.doi.org/10.1006/jpdc.2000.1668.

[66] Andi Toce, Abbe Mowshowitz, Akira Kawaguchi, Paul Stone, Patrick Dantressangle, and Graham Bent. HyperD: analysis and performance evaluation of a distributed hypercube database. In *The Annual Conference of the International Technology Alliance 2012 (ACITA'12)*, Botley Park Hotel, Southampton, UK, United Kingdom, September 2012.

[67] G. Bent, D. Dantressangle, A. Mowshowitz, and V. Mitsou. A dynamic distributed federated database. In *Proceedings of the Second Annual Conference of ITA*, 2008.

[68] G. Bent, D. Dantressangle, P. Stone, D. Vyvyan, and A. Mowshowitz. Experimental evaluation of the performance and scalability of a dynamic distributed federated database. In *Proceedings of the Second Annual Conference of ITA*, 2008.

[69] Paul Stone, Patrick Dantressangle, Graham Bent, Abbe Mowshowitz, Andi Toce, and Boleslaw K Szymanski. Query propagation behaviour in gaian database networks. In *The Annual Conference of the International Technology Alliance 2012 (ACITA'12)*, Botley Park Hotel, Southampton, UK, United Kingdom, September 2012.

[70] Paul Stone, Patrick Dantressangle, Graham Bent, Abbe Mowshowitz, Andi Toce, and Boleslaw K Szymanski. Query execution and maintenance costs in a dynamic distributed federated database. In *The Annual Conference of the International Technology Alliance 2012 (ACITA'12)*, Botley Park Hotel, Southampton, UK, United Kingdom, September 2012.

[71] Youyao Liu, Jungang Han, and Huimin Du. A hypercube-based scalable interconnection network for massively parallel computing. *JCP*, 3(10):58–65, 2008.

[72] Ümit V. Çatalyürek, Erik G. Boman, Karen D. Devine, Doruk Bozdag, Robert T. Heaphy, and Lee Ann Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *IPDPS*, pages 1–11, 2007.

[73] 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. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/606272.606299.

[74] Michael E. Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2011.

[75] Melissa S Cline, Michael Smoot, Ethan Cerami, Allan Kuchinsky, Nerius Landys, Chris Workman, Rowan Christmas, Iliana Avila-Campilo, Michael Creech, Benjamin Gross, Kristina Hanspers, Ruth Isserlin, Ryan Kelley, Sarah Killcoyne, Samad Lotia, Steven

Maere, John Morris, Keiichiro Ono, Vuk Pavlovic, Alexander R Pico, Aditya Vailaya, Peng-Liang Wang, Annette Adler, Bruce R Conklin, Leroy Hood, Martin Kuiper, Chris Sander, Ilya Schmulevich, Benno Schwikowski, Guy J Warner, Trey Ideker, and Gary D Bader. Integration of biological networks and gene expression data using cytoscape. *NATURE PROTOCOLS*, 2(10):2366–2382, 2007. ISSN 1745-2473. URL http://dx.doi.org/10.1038/nprot.2007.324.

[76] Azzedine Boukerche, Mohammad Z. Ahmad, Damla Turgut, and Begumhan Turgut. A taxonomy of routing protocols for mobile ad hoc networks. In Azzedine Boukerche, editor, *Algorithms and Protocols for Wireless Sensor Networks*, chapter 5, pages 129–164. Wiley Interscience, John Wiley Sons, Inc., 2009.

[77] Azzedine Boukerche, Daniel Camara, Antonio A.F. Loureiro, and Carlos M.S.Figueiredo. Algorithms for mobile networks. In Azzedine Boukerche, editor, *Algorithms and Protocols for Wireless Sensor Networks*, chapter 1, pages 1–20. Wiley Interscience, John Wiley Sons, Inc., 2009.

[78] Marcel Castro, Andreas Kassler, Carla-Fabiana Chiasserini, Claudio Casetti, and Ibrahim Korpeoglu. Peer-to-peer overlay in mobile ad-hoc networks. In Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 1045–1079. Springer Publishing Company, Incorporated, 2009.

[79] Elizabeth M. Royer and C.-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6:46–55, 1999.

[80] David B. Johnson, David A. Maltz, and Josh Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pages 139–172. Addison-Wesley, 2001.

[81] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, 2003.

[82] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, INFOCOM '97, pages 1405–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7780-5. URL http://dl.acm.org/citation.cfm?id=839292.843010.

[83] Chai-Keong Toh. Associativity-based routing for ad hoc mobile networks. *Wirel. Pers. Commun.*, 4(2):103–139, March 1997. ISSN 0929-6212. doi: 10.1023/A:1008812928561. URL http://dx.doi.org/10.1023/A:1008812928561.

[84] Rohit Dube, Cynthia D. Rais, Kuang yeh Wang, and Satish K. Tripathi. Signal stability based adaptive routing (ssa) for ad-hoc mobile networks. *IEEE Personal Communications*, 4:36–45, 1997.

[85] Mesut Gnes, Udo Sorges, and Imed Bouazizi. Ara - the ant-colony based routing algorithm for manets, 2002.

[86] J. J. Garcia-Luna-Aceves, Marc Mosko, and Charles E. Perkins. A new approach to on-demand loop-free routing in ad hoc networks. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pages 53–62, New York, NY, USA, 2003. ACM. ISBN 1-58113-708-7. doi: 10.1145/872035.872043. URL http://doi.acm.org/10.1145/872035.872043.

[87] Prince Samar, Student Member, Marc R. Pearlman, Zygmunt J. Haas, and Senior Member. Independent zone routing: An adaptive hybrid routing framework for ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 12:595–608, 2004.

[88] Guangyu Pei, Mario Gerla, and Tsu wei Chen. Fisheye state routing: A routing scheme for ad hoc wireless networks. In *IN PROCEEDINGS OF ICC 2000*, pages 70–74, 2000.

[89] Mario Gerla, Xiaoyan Hong, and Guangyu Pei. Landmark routing for large ad hoc wireless networks, 2000.

[90] George Aggelou and Rahim Tafazolli. Rdmar: a bandwidth-efficient routing protocol for mobile ad hoc networks. In *Proceedings of the 2nd ACM international workshop on Wireless mobile multimedia*, WOWMOM '99, pages 26–33, New York, NY, USA, 1999. ACM. ISBN 1-58113-129-1. doi: 10.1145/313256.313272. URL http://doi.acm.org/10.1145/313256.313272.

[91] Guoqiang Wang, Yongchang Ji, and Dan C. Marinescu. A routing protocol for power constrained networks with asymmetric links. In *in Proceedings of the ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN*, pages 69–76, 2004.

[92] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24 (4):234–244, October 1994. ISSN 0146-4833. doi: 10.1145/190809.190336. URL http://doi.acm.org/10.1145/190809.190336.

[93] Optimized link state routing protocol (olsr), 2003.

[94] Ching chuan Chiang and Mario Gerla. Routing and multicast in multihop, mobile wireless networks. In *in Multihop, Mobile Wireless Networks, in Proc. IEEE ICUPC '97*, 1997.

[95] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mob. Netw. Appl.*, 1(2):183–197, October 1996. ISSN 1383-469X. doi: 10.1007/BF01193336. URL http://dx.doi.org/10.1007/BF01193336.

[96] J.J. Garcia-Luna-Aceves and Marcelo Spohn. Source-tree routing in wireless networks. pages 273–282, 1999.

[97] R. Ogier, F. Templin, and M. Lewis. Topology dissemination based on reverse-path forwarding (tbrpf), 2004.

[98] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wirel. Netw.*, 6(4):307–321, July 2000. ISSN 1022-0038. doi: 10.1023/A:1019106118419. URL http://dx.doi.org/10.1023/A:1019106118419.

[99] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (dream). In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '98, pages 76–84, New York, NY, USA, 1998. ACM. ISBN 1-58113-035-X. doi: 10.1145/288235.288254. URL http://doi.acm.org/10.1145/288235.288254.

[100] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM. ISBN 1-58113-197-6. doi: 10.1145/345910.345953. URL http://doi.acm.org/10.1145/345910.345953.

[101] Jian Li and Prasant Mohapatra. Laker: learning from past actions to guide future behaviors in ad hoc routing: Research articles. *Wirel. Commun. Mob. Comput.*, 7(4):495–511, May 2007. ISSN 1530-8669. doi: 10.1002/wcm.v7:4. URL http://dx.doi.org/10.1002/wcm.v7:4.

[102] Giulia Boato, Giulia Boato, Fabrizio Granelli, and Fabrizio Granelli. Mora: a movement-based routing algorithm for ad hoc networks.

[103] Alvin Valera, Winston Seah, S.V. Rao, and Sv Rao. Cooperative packet caching and shortest multipath routing in mobile ad hoc networks. In *in Proceedings of IEEE INFOCOM, March-April 2003*, pages 260–269, 2003.

[104] Mahesh K. Marina and Samir R. Das. On-demand multipath distance vector routing in ad hoc networks. In *in Proceedings of IEEE International Conference on Network Protocols (ICNP*, pages 14–23, 2001.

[105] Sung ju Lee and Mario Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks, 2001.

[106] Theodore Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2001. ISBN 0130422320.

[107] Mario Gerla and Jack Tzu chieh Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1:255–265, 1995.