Peer-to-peer Network Based on the Knödel Graph

Junlei He

A thesis

in

The Department

of

Computer Science

Presented in partial Fulfilment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2007

Your file Votre référence
ISBN: 978-0-494-28951-8
Our file Notre référence
ISBN: 978-0-494-28951-8

Abstract

Peer-to-peer network based on the Knödel graph

Junlei He

Knödel graphs $W_{d,n}$ of even order n and degree d, $1 \le d \le \lfloor \log_2 n \rfloor$, was introduced by W. Knödel for broadcasting and gossiping information in interconnected networks. It has been proved that the diameter of the Knödel graphs $W_{d,2^d}$ is $\lceil \frac{d+2}{2} \rceil$.

In recent years, peer-to-peer (P2P) computing has attracted a lot of attention from the researchers and application developers. With the development of mobile networks, mobile peer-to-peer (MP2P) computing may become a potential "killer application" of the mobile carriers.

In this thesis, we present a P2P network structure based on the Knödel graphs. The routing algorithm takes advantage of the routing heuristics in the Knödel graphs and reduces the routing length of P2P network by half, compared with the currently studied P2P network structures (CHORD and CAN). Given that most mobile devices have limited computing resources, another advantage of the proposed P2P network structure is that it can be flexibly configured to adapt to different MP2P application scenarios, taking into the fact that most mobile devices have limited computing resources. With this model, each mobile node maintains a fixed-size routing table regardless of the size of the whole network, while the tradeoff on routing hops is still acceptable in many MP2P applications.

Some practical heuristics for real mobile network are also proposed in this thesis.

Acknowledgements

I truly appreciate the help of my supervisor, Dr. Hovhannes A. Harutyunyan, one who has guided me along since the very beginning. His charisma has inspired me throughout the course of my learning, and will continue to do so.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Definition of Peer-to-Peer Computing

Popularized by Napster ([1]) and Gnutella ([2]) file sharing solutions, peer-to-peer (P2P) computing has attracted much attention of the researchers and developers. The initial purpose of these P2P systems was to share huge volumes of data. With the improvement of network bandwidth and processing abilities of most nodes, some P2P applications aimed sharing resources other than the contents of each node in the network, such as hardware resource (CPU cycles, storage space, etc). Collaborative computing and communication are also treated as P2P application in some scope (e.g.: instant messaging).

In fact, P2P applications are not a new idea. Although consumer applications for P2P are relatively recent, the concept of P2P is as old as the Internet itself. The P2P nature of the early Internet was best exemplified by the Usenet network. Created in 1979, Usenet is a network of computers (accessed via the Internet), each of which hosts the entire contents of the network. Messages are propagated between the peer computers; users connecting to any single Usenet server have access to all the messages posted on each individual server. While the users' connection to the Usenet server is of the traditional client/server model, the relationship between servers is definitely P2P.

Until now, there is not a widely accepted definition of peer-to-peer computing. In general, the notion of P2P means that the processing is spread over a large number of "agents" (servents, in Gnutella) with minimal central control. Different from the traditional client/server model (which still dominates in network), in a P2P system, usually but not necessarily, peers function as both client and server and have equivalent responsibilities in the network.

Comparing with the widely accepted client/server computing model, P2P systems have some advantages such as:

- May take advantage of many unused resources in the internet;
- Avoid the computing ability and bandwidth bottleneck of a central server;
- Improve the system robustness;
- Increase privacy;
- Result in more flexible and adaptable networks, etc.

On the other hand, P2P systems also have some drawbacks:

- There is no guarantee that the required content/resources will always be available;
- It is hard to enforce content ownership in P2P network;
- The efficiency of P2P systems is a problem, including bandwidth usage and routing strategies;
- P2P systems may be more vulnerable to viruses and attacks, etc.

P2P applications also have some other potential influence on Internet. Current technologists and users tend to forget that the Internet is meant to facilitate something more like a conversation and less like a television network. P2P redresses some of this imbalance between user and provider and makes use of more participants than spectators.

With the popularity of the wireless communication network, such as WLAN and 3G networks, some mobile peer-to-peer (MP2P) applications are becoming the so-called killer applications. After the mobile IP (RFC3344, [32]) is practically in use, the mobile node may keep the same IP address when it moves around different networks. This makes it possible for the mobile nodes to participate in the P2P applications without modifying the existing P2P protocols. Some researchers have already done experiments to prove that, by treating the mobile nodes as ordinary network nodes, P2P file sharing (eDonkey [35], in the experiments) could be mapped onto the GPRS and UMTS networks ([33], [34]). We will discuss the issues brought by the mobility in later sections.

1.2 Peer-to-peer Applications and Limitation on Mobile Environment

In this section we categorize P2P applications and provide examples for each category.

1.2.1 Content Sharing

Content sharing is still the dominant application in P2P network. After Napster and Gnutella, many other P2P content sharing systems were introduced to Internet users, such as Freenet ([3]), KaZaA ([4]), BitTorrent, etc. Each of the systems has its own characteristics, some of them focused on locating content, while others dedicated to improving content spreading efficiency.

1.2.2 Other Resource Sharing

Besides data contents, other computing resources can also be shared in a P2P manner. Proposed by Microsoft and Starbucks, SFLAN (an experimental wireless community network in the San Francisco Bay Area, [6]) is an example of using P2P as a shared resource provider. The basic idea behind SFLAN is that, as the use of broadband connections grows, people will create LANs in their houses using small, inexpensive hubs into which they can plug their computers. They will also be able to plug in a tiny radio beacon (a WAP) so that anyone within range can piggyback on his/her broadband Internet connection. Eventually, this creates a citywide equivalent of a multimegabit-per-second LAN for the benefit of all at a very small cost of each user.

1.2.3 Instant Messaging

Instant messaging (IM) systems have been around for many years. The IM systems, such as ICQ, AOL, Yahoo Messenger and MSN Messenger, are very popular today. Jabber ([5]) is believed to be the next generation of IM platform, which is unique in that it is also a platform for building other kinds of applications.

1.2.4 Distributed Computing

The "Search for Extraterrestrial Intelligence at Home" (SETI@home, [8]) project is an example of distributed computing. It makes use of idle computers in the Internet to analyze the data collected continuously from large radio telescopes at Arecibo and other places in the world. The SETI@home server breaks down the whole task into small work units and sends them to computers all over the world. When the computer finishes one job, it sends the result back and gets the next work unit. Climate prediction is another distributed computing application that attracts a lot of attention.

1.2.5 Group Collaboration

Groove ([7]) is a platform that software developers can use to build a cooperative collaboration environment. The team members install Groove on PCs and create a "virtual space". In this virtual space, one member can interact with other members to collaborate on projects in real time as if they were both in the same room.

Some other applications are also of P2P style, which share similar system architectures with some of the systems mentioned above. For example, some online games are similar to IM systems, and some enterprise distributed storing systems are similar to content sharing systems.

Most of these applications may also run in a mobile environment. Some of them may even become the new attraction for the mobile network operators. For example, the mobile online gaming is definitely more attractive to the players than that running on stationary desktops.

1.3 Research Concerns about P2P and MP2P Computing

The research on P2P systems may involve many aspects, including network architecture, querying, routing, caching strategies, mapping with the underlying physical network, business model, privacy and security, etc. As a new technology, many of these topics deserve a lot of research.

Compared with the traditional network, the mobile network system has many limitations including, but not limited to:

- The connection status is unreliable.
- The bandwidth of the mobile network is lower and more expensive.
- The mobile devices usually have limited computing capability.
- Energy efficiency is a critical issue for these devices.

These limitations bring more issues for the mobile peer-to-peer (MP2P) computing. The following sections describe the studies in more details.

1.3.1 P2P Network Architecture

A P2P network can be modeled as a graph $G = (V, E)$, where V is the set of vertices (or peers) and E is the set of edges (or communication links between peers). Two peers $u \in V$ and $v \in V$ are *adjacent* if there is an edge $e \in E$, such that $e=(u, v)$. We also say peers u and v are *neighbours* of each other. The *degree* of a peer is the number of neighbours of this peer. The *degree* of a graph G is the maximum degree among all peers in this graph. Δ stands for the degree of a graph. A path p in a graph $G = (V, E)$ is a sequence of peers of the form $p=v_1, v_2, ..., v_n$ $(n>1)$, in which each peer v_i is adjacent to the next node v_{i+1}. Obviously, the path p is also a sequence of edges. The length of a path is the number of edges in the path. The length of the shortest path between two peers is the *distance* between them. The *diameter* of a graph is the maximum of the distances between all pairs of peers in the graph. A graph $G = (V, E)$ is said to be *connected* if there is a path between any two peers on G.

In P2P networks, large diameter means that querying and responding messages have to travel long distances (hops) and might be forwarded to more irrelevant peers. This causes low efficiency in both response time and bandwidth usage. In addition, high degree means that each peer has to deal with a large routing table, and when peers join or leave the network, more peers have to be notified. Thus high degree leads to high maintenance overhead and bad scalability. Different from interconnection networks in parallel computers, peers' random joining and leaving the network make it difficult to adopt the architectures (such as *hypercube*, *DeBruijn* graph, etc. [9]) widely used in interconnection networks.

Early P2P networks, such as Gnutella, are formed arbitrarily. A new peer randomly chooses some peers in the network and connects to them. When a peer leaves the network, its neighbours just simply drop the links to it. This strategy makes the maintenance of the network simple; however, it also leads to large diameter of the network and high degree for those popular peers, which implies bad scalability. Some newly developed P2P network models are trying to address this problem by arranging the network peers in certain organized way to reduce the diameter and degree of the whole network. *CHORD* ([10]) and *CAN* ([11]) are two such P2P models. In *CHORD*, with high probability, the diameter and degree of the network are both of $O(logN)$ (N is the total number of peers). In CAN, the average routing path length is $(d/4)(n^{1/d})$ and an individual peer maintains $2d$ neighbours (*CAN* partitions a d dimensional space into n equal zones). More details of these two P2P models will be introduced in chapter 2.

In this thesis, we propose a new MP2P network architecture in which the number of degree is configurable between a constant number and log$N$ according to different application scenarios.

1.3.2 Querying and routing strategy

The main purpose of a P2P network is to locate and retrieve certain resources. The search result is usually evaluated by the costs and the quality of the results. Costs include bandwidth usage and processing cost of involved peers. Some P2P systems, such as Gnutella, use a simple flooding strategy to broadcast the originator's querying requirement, and limit the hops each message will travel in the network by using a parameter called *Time-To-Live* (TTL). However, this strategy does not guarantee that the desired content will be retrieved, even if it does exist in the network. Also, this strategy needs too much bandwidth of the network. Researchers are developing some other strategies to accomplish the P2P querying, such as *iterative deepening, directed BFS, local indices, hints*, etc ([12], [13]).

Besides the exact matching query, range query is also studied by researchers ([15], [16]).

1.3.3 Network mapping

Most P2P networks are application-level systems on top of the Internet. Each link in P2P network may span several physical hops of TCP/IP link. Figure1 illustrates how the mismatching influences the performance of P2P networks.

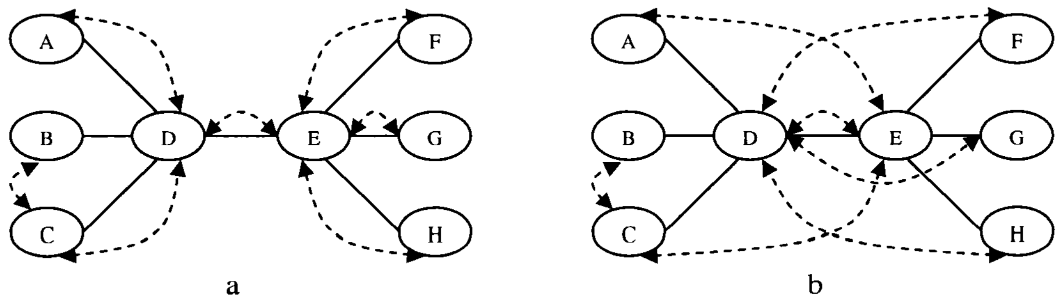

Figure 1 – Mapping P2P and the Internet (Solid lines stand for underlying TCP/IP network links, dotted lines stand for P2P network links.)

In the perfect mapping P2P network (Figure 1.a), when peer *A* broadcasts a message to all the other 5 peers, the message travels the physical link *D-E* only once. In the inefficient P2P network (Figure 1.b), the same operation will cause the message to travel across link *D-E* 6 times. This example shows that the mismatch between P2P overlay network topology and the Internet infrastructure has critical performance implication. In the MP2P environment, this is a serious problem if the *D-E* link is a low bandwidth or expensive link (e.g.: wireless interface).

Reconfiguring P2P network to map the underlying TCP/IP network perfectly is quite difficult or even impossible. However, some change to the P2P protocol may help to improve the mismatching problem, thus achieve better efficiency and scalability ([17]).

With mobile IP, there may be additional network mapping issues. According to the mobile IP protocol, when a mobile node sends a packet to a corresponding node (*CN*) in the Internet, it has two options: one is to send the packet directly to the *CN* using its *care-of-address* (*CoA*), another is to send the packet back to its home agent (*HA*) and the *HA* sends the packet to the *CN* using its home address. In the first option, the *HA* intercepts the packets from the *CN*, and then forwards the packets to the mobile node in a tunnel. The path of the incoming and outgoing traffic forms a triangle. Usually this option is more efficient regarding the outgoing traffic. However, the *CN* must be aware that the *CoA* belongs to the specific node which is participating in a P2P session. Furthermore, the *CoA* is subject to change when the mobile node moves to another network. The second option avoids the *CoA* recognition problem, but the transmission efficiency is affected if we do not carefully consider the network mapping strategy.

1.3.4 Finding good peers

Although peers have equivalent responsibility in a P2P network, differences do exist from peer to peer. Some peers may have wider bandwidth and/or better processing ability, while some other peers may have more resources to share. Finding a good peer to connect to, or selecting a peer that has the required resource (there may be more than one peer having the resource), can definitely improve the performance of the P2P system. By adding some information into the message pack, peers can figure out where the "good" peers are located ([18], [19]).

In a hybrid MP2P network which contains both mobile nodes and stationary nodes, the stationary ones are more likely to have wider bandwidth and their status is more stable. To make the system more robust, it is preferred to select these stationary nodes to form the backbone (or say, core) of the network.

1.3.5 Business model, Privacy, Security, Trust and Incentives

One major problem with P2P computing is how to protect the authority of information owner. On the other hand, when authority is not a problem (such as in Instant Messaging), how to make money from the platform is a big concern. Researchers have proposed approaches so that the authors can get profit according to how many times their music has been downloaded. Other discussions include pushing advertisements to specific people who may be interested, and even online gambling, etc ([20], [21]).

1.4 The New P2P Network Architecture

In this thesis, we propose a new P2P network architecture and related algorithms. The architecture is based on the Knödel graph ([26]). When using the partial Knödel graph, the constant degree implies that each peer is connected to a fixed number of other peers, thus when peers join or leave the network, the processing overhead is constant and will not increase when the size of the network increases. This feature is especially good for many mobile devices that have limited computing resources. While configured with degree of log$N$, the diameter of the network is approximately half of that in CHORD.

1.5 Thesis outline

The remaining of this thesis is structured as follows:

Chapter 2 introduces the existing work done on P2P network architecture. We mainly explain the two most cited models: CHORD and CAN.

In Chapter 3, we propose a new P2P network architecture based on Knödel graph. We present the routing and maintaining algorithms for the new architecture. The last section of this chapter introduces some practical proposals for P2P computing in the real mobile network with mobile IP.

Simulation results are introduced in Chapter 4. The results are also compared with that of CHORD and CAN.

Chapter 5 concludes the thesis and lists the future work.

2 Related Work

In this chapter, we first introduce the Gnutella network, which is the basis for many current P2P researches. After that, we will introduce two new representative P2P network models: CHORD and CAN. Several other P2P network models (Freenet and GRAPES) will be introduced in the last section of this chapter.

2.1 Gnutella

Different from Napster ([1]), which is considered to be the first large scale P2P network, Gnutella ([2]) is totally decentralized - it does not need any central directory. Content sharing P2P networks tend not to use a centralized directory since Napster met lawsuit trouble. The distinction of Gnutella protocol is its decentralized model, although Gnutella supports traditional client/centralized server search paradigm. In Gnutella, each peer acts as both a server and a client, and is called a *servent*.

Gnutella defines a set of descriptors used for communicating data between servents and a set of rules governing the inter-servent exchange of descriptors. The defined descriptors include:

- Ping – Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
- Pong – The response to a Ping. It includes the address of a connected Gnutella servent and information of the amount of data it is making available to the network.

- Query – The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found in its local data set.
- QueryHit – The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
- Push – A mechanism that allows a firewalled servent to contribute the data files to the network.

A Gnutella servent connects itself to the network by establishing a connection with another servent currently on the network. Once the address of another servent on the network is obtained, a TCP connection to the servent is created. The Gnutella connection is created after a series of string exchanges.

A well-behaved Gnutella servent will route protocol descriptors according to the following rules:

- Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor;
- QueryHit descriptors may only be sent along the same path that carried the incoming Query descriptor;
- Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptor;
- A servent will forward incoming Ping and Query descriptors to all of its directly connected servents, except the one that delivered the incoming Ping or Query. This is the *M2* Model of *Messy broadcasting* ([22]);

- A servent will decrease a descriptor header's TTL field, and increase its Hops field, before it forwards the descriptor to any directly connected servent;
- If a servent receives a descriptor with the same Payload Descriptor and Descriptor ID as the one received before, it should attempt to avoid forwarding the descriptor to any connected servent.

Once a servent receives a QueryHit descriptor, it may initiate the direct download of one of the files described by the descriptor's Result Set. Files are downloaded out of Gnutella network, i.e., the file is not transferred along the QueryHit descriptor's route. A direct connection between the source and target servent is established in order to perform the data transfer. The file download protocol is HTTP.

2.2 CHORD

Gnutella is a nondeterministic P2P network, i.e., it does not guarantee that a search will retrieve a successful result even if the desired content does exist in the network. CHORD ([10]) and CAN([11], introduced in next section) solved this problem by maintaining a hashed content directory and defining certain network architecture and related routing strategies. In this section, we briefly introduce how CHORD works. CAN will be introduced in the next section.

2.2.1 Routing in CHORD

The CHORD protocol specified how to find the locations of keys, how peers join and leave the system, and how to recover from the failure of existing peers. CHORD predefines a large number of virtual nodes for the whole system (the number should be large enough to make the possibility that two real peers have to share a virtual node very low), and assigns each virtual node an identifier. The virtual node space is arranged into a circle. Each real peer in the network is in charge of a section on the circle (or some continuous virtual nodes on the circle). The position of each peer is determined by hashing its IP address using *consistent hashing* ([24]). The locations of the contents are hashed to keys and are maintained by the virtual nodes. When searching for certain content, the client use the hashing function to calculate which virtual peer should be maintaining the location information of the desired content, and then forward a querying message to the real peer that is in charge of that specific virtual node.

Suppose the CHORD system is using m number of bits in the key/peer identifiers. Each peer, n, maintains a routing table (or finger table) with at most m entries. The i^{th} entry in the table at node n contains the identity of the first node, s, that succeeds n by at least 2^{i-1} on the identifier circle, i.e., $s=successor(n+2^{i-1})$, where $1 \leq i \leq m$ (all arithmetic is modulo 2^m). A routing table entry includes both the CHORD identifier and the IP address (and port number) of the relevant peer. It is easy to see that the first finger of n is its immediate successor on the circle.

When a peer n does not know the successor of key k, n searches its routing table for the node j whose ID most immediately precedes k, and forwards the querying message to j. By repeating this process, n learns about peers with IDs closer and closer to k. and finally the querying message will reach peer k.

2.2.2 Maintaining CHORD Network

Peer Joining

A new peer n learns the identity of an existing CHORD peer n' by some external mechanism. Peer n initializes its state and adds itself to the network as follows.

- Initializing the routing table and predecessor. According to the IP address of *n*, *n'* may know the final position of *n*. By finding successor for each entry, *n*'s routing table can be initialized. Some practical optimization may reduce the time to fill the routing table to *O(log N)*, where *N* is the total number of peers in the network.
- Updating the routing table of existing peers. The new peer, *n*, will become the i^{th} finger of peer *p* if and only if: (1) *p* precedes *n* by at least 2^{i-1}, and (2) the i^{th} finger of node *p* succeeds *n*. When *n* joins, each peer of the situation above must update its own routing table respectively. The number of such peers is *O(log N)* with high probability (the phrase "with high probability" is based on the standard hardness assumptions to decrypt SHA-1 algorithm, see [10] for more detail). Finding and updating these peers takes $O(\log^2 N)$ time, and some sophisticated scheme can reduce this time to *O(log N)*.
- Transferring keys. The last step is to move the keys that should now be maintained by *n*. The relevant keys were stored in the predecessor of *n*. Thus *n* only needs to contact one peer to transfer the keys.

Peer Leaving and Failure

When a peer leaves the CHORD network as scheduled, basically it performs the reversed steps as those for joining, except the step of initializing the local routing table. However, if a peer fails, there will be some more work to do to recover the system.

The key step in failure recovery is maintaining correct successor pointers. Thus beside the routing table, each CHORD peer maintains a list of successors of its *r* nearest successors on the CHORD circle. If node *n* notices that its successor has failed, it replaces it with the first live entry in its successor list. After some time, a certain stabilizing algorithm defined in CHORD will correct routing table entries and successor-list entries pointing to the failed peer.

Study ([10]) shows that if the length of the successor list, *r*, is of *O(log N)*, and every peer fails with probability 1/2, then with high probability, the closest living successor to the query key will be returned, and the expected time to find it in the failed network is of *O(log N)*.

According to our observation, some key and/or data replication should be done to improve querying quality when peers fail.

2.3 CAN

The Content Allocatable Network (CAN) treats the peer space as a virtual *d*-dimensional Cartesian coordinate space on a *d*-torus. Key *K* is deterministically mapped onto a point *P* in the coordinate space using a uniform hash function. The corresponding *(key, value)* pair is then stored at the peer that owns the zone within which the point *P* is located. To retrieve an entry corresponding to key *K*, any peer can apply the same deterministic hash function to map *K* onto point *P* and then retrieve the corresponding value from the point *P*. If the point *P* is not owned by the requesting peer or its immediate neighbours, the request must be routed through the CAN infrastructure until it reaches the peer in whose zone *P* belongs to.

2.3.1 Routing in CAN

Simply to say, routing in CAN works by following the straight line path through the Cartesian space from the source to destination coordinates. A CAN peer maintains a coordinate zone of each of its immediate neighbours in the coordinate space. In a d-dimensional coordinate space, two peers are neighbours if their coordinates span overlap along d-1 dimensions and abut along one dimension.

For a d-dimensional space partitioned into N equal zones, the average routing path length is $(d/4)(N^{1/d})$ hops and individual peer maintains $2d$ neighbours. When d=log$N$, the average routing length is of O(log$N$). However, the value of d must be determined before constructing the network, and will not change when the size of the whole network (N) changes.

2.3.2 Peer Joining

A new peer joins the CAN by the following steps.

1. Find a peer already in the CAN (Bootstrap). The CAN protocol does not specify what kind of bootstrapping mechanism should be employed. Any method may be used to acquire the IP address of an existing peer in the CAN.
2. Using CAN routing mechanisms, find a peer whose zone will be split. The bootstrap peer supplies the IP addresses of several randomly chosen peers currently in the system. The new peer randomly chooses a point P in the space and sends a JOIN request destined for point P. This message is sent into the CAN via any existing CAN peer.
3. Notify the neighbours of the split zone, update their routing table. The new peer learns the IP addresses of its coordinate neighbour set from the previous occupant. The neighbours of the previous occupant, and the occupant itself, update their routing table accordingly. The number of peers involved in the

splitting depends only on the dimensionality of the coordinate space and is independent of the total number of peers in the system. This means that the splitting time is almost constant. After the zone is splitted, the previous occupant transfers part of the *(key, value)* pairs to the new peer.

2.3.3 Peer Departure, Recovery and Maintenance

Scheduled peer departure can be easily handled, the departing peer transfers the (*key, value*) pairs to the peer that will take charge of its zone, and notifies its neighbours so that they can update their routing table.

CAN employs an "immediate takeover algorithm" to ensure one of the failed peer's neighbours takes over the zone. In this case the (*key, value*) pairs held by the failed peer are lost until the state is refreshed by the holders of the data. To prevent stale entries as well as to refresh lost entries, peers that insert (*key, value*) pairs into the CAN periodically check and refresh these entries (if needed).

2.4 Others: Freenet, GRAPES...

Similar to Gnutella, Freenet ([3]) uses a fully distributed model but also introduces some innovations (anonymity, caching, etc). It employs depth-first-search (DFS) with depth limit D (Gnutella's search strategy is breadth-first-search). Another difference between Freenet and Gnutella is that, in Freenet, the requested object is returned backwards along the request path, and each peer in the path caches the response object in order to satisfy future requests more quickly. This also provides some degree of anonymity for the source of the requested object. On the other hand, this mechanism will need more bandwidth and storage space, and more delay will occur when retrieving the requested object in some cases.

Provided that the central server has enough bandwidth and processing ability, a centralized directory is obviously more efficient than a decentralized one. However, some non-technical facts may prevent the centralized directory from being employed in an application system, such as financial problems, legal problems, etc. Therefore, some researchers have proposed with the idea of maintaining a content directory that is between centralized and fully decentralized to address certain requirements. GRAPES ([25]) is such an experimental system. It has a two-layer hierarchical structure. The peers are organized into many sub-networks; each sub-network elects a super node to form the super-network. The super-nodes maintain the content directory of the whole system. Compared with the systems with plain structure (such as CHORD and CAN), GRAPES may halve the querying time, at the cost of increased overhead to maintain the 2-layer structure. The cost may be compensated somehow by clustering the nearby peers together.

3 The New P2P Infrastructure and Related Protocol

The main task for a P2P network is to locate certain resources in the network and then retrieve data from the corresponding node or send information to it. In many cases, the second part can be accomplished directly within the underlying TCP/IP or some other network infrastructure, as long as the IP address of destination node is found with the P2P protocol. However, in some content sharing P2P networks, certain hot content might be cached in many intermediate nodes to speed up the downloading. Many researchers developed several methods for the caching scheme. In our research, we concentrate on the first part, i.e., modeling the P2P network and defining related querying and routing algorithms.

Without the presence of a centralized server node, maintaining the distributed resource directory and locating the desired resource differ a lot from that in a client/server system. In P2P network, every node acts equally, or almost equally, as both client and server. And the nodes may join or leave the network randomly at any time, while the performance of the network is not affected significantly. On the contrary, the server of a client/server system should never be out of service otherwise the whole network is stalled.

3.1 The P2P Architecture Based on the Knödel Graph

Knödel graphs ([26]) have many good properties in terms of network broadcasting and gossiping ([27]). It was first proposed by W. Knödel in 1975 as a gossiping structure ([26]), and was formally defined in [28] as follows.

Definition 1 [28] (Knödel graph)

The Knödel graph on $n \geq 2$ vertices (n even) and of maximum degree d, $1 \leq d \leq \lfloor \log_2 n \rfloor$ is denoted by $W_{d,n}$. The vertices of $W_{d,n}$ are the pairs (i, j) with $i = 0,1$ and $0 \leq j \leq \frac{n}{2} - 1$, and the set of edges:

$$E = \left\{ ((0,i),(1,j)) \mid j = i + 2^r - 1 \bmod \frac{n}{2}, 0 \leq i, j \leq \frac{n}{2} - 1,\ 0 \leq r \leq d-1 \right\} \tag{2.1}$$

For $0 \leq k \leq d-1$, an edge of $W_{d,n}$ that connects a vertex (0, j) to the vertex (1, j+2^k-1 mod n/2) is said to be in *dimension k.* It is obvious to see that the Knödel graph contains a Hamiltonian cycle which is formed by alternating edges in dimension 0 and 1.

Figure 2 shows the Knödel graph $W_{3,12}$.

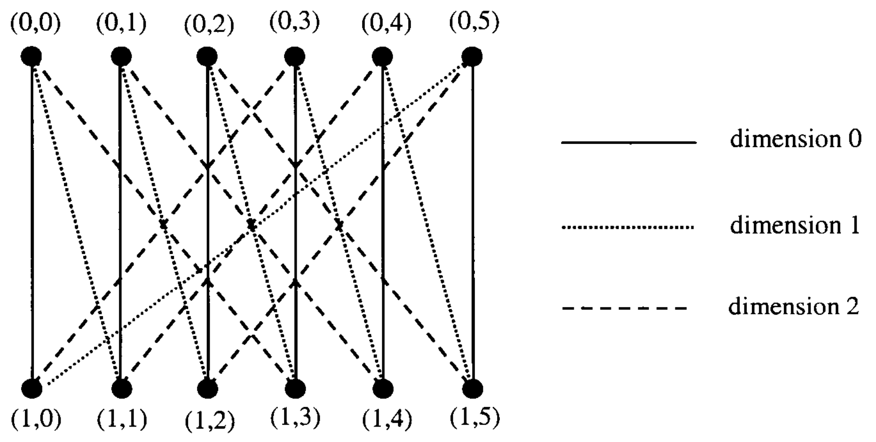

Figure 2 - Knödel graph of dimension 3 and order 12

Study ([23]) shows that in many cases, such as $W_{k,2^k}$ and $W_{k-1,2^{k-1}}$, Knödel graphs are minimum broadcast graphs and minimum gossip graphs.

There are other definitions of Knödel graph, such as 1-layer representation, definition of Knödel graph as Cayley graphs, etc. [27] and [29] have more details about these definitions.

Comparing Knödel graph $W_{d,2^d}$ with the underlying infrastructure of CHORD of the same order 2^d, we can see that they share the same degree d and their diameters are of the same order - $\lceil \frac{d+2}{2} \rceil$ and d respectively ([30]). We can see that Knödel graph $W_{k,2^k}$ has smaller diameter compared to the underlying graph in CHORD. Next we will introduce several routing algorithms of the Knödel graph: a simple routing, a routing algorithm for the partial Knödel graph with limited dimensions, and the heuristics for the routing in a full Knödel graph $W_{d,2^d}$.

3.1.1 Basic Routing in Knödel Graph

Simple Routing

As mentioned before, edges of dimension 0 and 1 form one Hamiltonian cycle of the Knödel graph. Thus the simplest routing is just to travel along the Hamiltonian cycle, or, in other words, use edge of dimension 0 and 1 alternatively, to reach the destination from the source.

In this case, the longest route has length N-1 and is not efficient. However, this algorithm is very simple, and it may be used in the last phases in the routing algorithms described below (explained in the following section).

Routing in the Partial Knödel Graph

A full Knödel graph $W_{d,2^d}$ is of degree *d*, i.e., each node is linked to *d* edges (dimension 0 to dimension *d*-1). We define a partial Knödel graph with degree less than *d*. It will have dimension 0, dimension 1, and some dimensions between 2 and *d*-1.

To illustrate the routing mechanism, we first consider a partial Knödel graph with 3 dimensions, dimension 0, dimension 1 and dimension *k*. The edges of dimension *k* ($1 < k \leq d-1$) are the chords inside the Hamiltonian cycle formed by dimension 0 and 1.

To simplify our discussion, we re-number the nodes in $W_{d,2^d}$ as follows:

$$\begin{cases} (0,i) \Rightarrow 2i \\ (1,j) \Rightarrow (2j-1) \bmod 2^d \end{cases} \tag{2.2}$$

After the re-numbering, the Knödel graph described above for *d*=4 can be drawn as in figure 3.

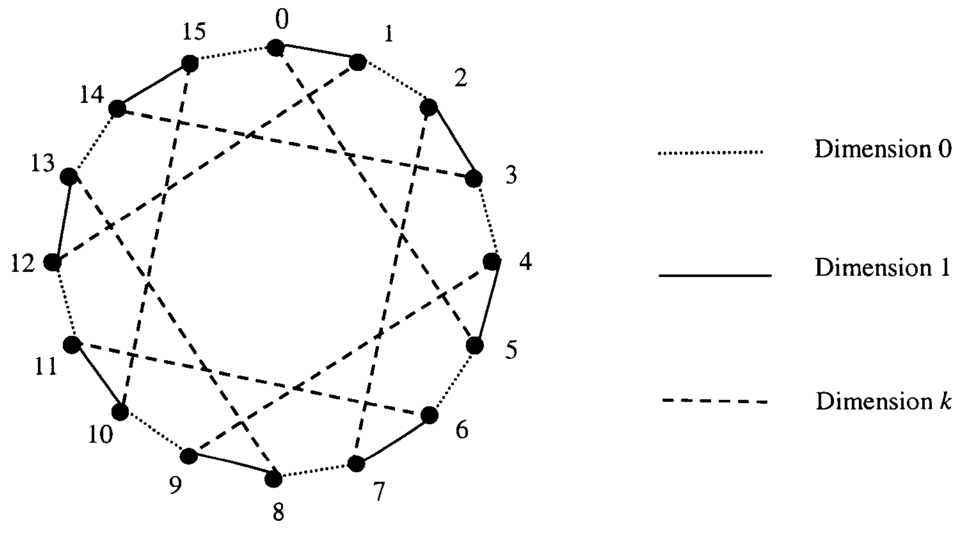

Figure 3 - Partial Knödel Graph $W_{4,16}$ of 3 Dimensions, *k*=2

From figure 3 we can observe that:

- Vertices of layer 0 (originally numbered as (0, *j*)) become even numbered vertices, and vertices of layer 1 (originally numbered as (1, *j*)) become odd numbered vertices. All the numbers are located continuously on the cycle.
- Each chord inside the cycle is an edge of dimension 2, and it spans 5 edges on the cycle. Simple calculations show that the number of edges the chord spans is not related to the total number of vertices in the graph, it is only related to which dimension we are choosing.
- The *k*-th dimension ($2 \leq k \leq d$-1) actually spans ((2^{k+1}-3) mod 2^d) edges on the circle. If we say that dimensions 2 to *d*-2 hop forward certain steps along the circle, dimension *d*-1 will actually hops 3 steps backward.

This observation leads to the following routing mechanism, which has a predictable routing length l ($l \leq \frac{N}{6} + 2$), where N is the total number of vertices in the graph ($N = 2^d$). Since Knödel graphs are vertex-symmetric, without loss of generality, we consider the routing problem starting from vertex 0. We divide the cycle into left half and right half by the axis consisted of vertex 0 and vertex 2^{d-1} (vertex 8, in figure 3), and then use the following routing steps:

- Determine which half the destination vertex is located in.
- If the destination is located in the right half, use edges of dimension 2 and dimension 1 alternatively. Every two hops may forward 6 edges alone the cycle closer to the destination. If the destination is located in the left half, use edges of dimension 0 and dimension 2 instead.

- When approaching close to destination, use edges of dimension 0 and 1 respectively to reach the destination along the circle.

When N is large, this routing length (of $O(N)$) is not good enough and may lead to scalability problem. Besides edges of dimension 0 and 1, which are necessary to form the Hamiltonian cycle, we may consider to use edges of higher dimension for the 3rd degree. It is easy to see that edges of higher dimension span more distance along the Hamiltonian cycle, thus it is faster to get near the destination by using these edges. However, it may take more steps to reach the destination in the last phase (traveling along the Hamiltonian cycle). To minimize the number of hops D without increasing the degree of the graph, the following calculation reveals the best dimension that should be used as the 3rd degree.

Suppose we are using the x-th dimension as the 3rd degree. The 3rd degree edges will span $(2^{x+1}-3) \bmod 2^d$ along the Hamiltonian cycle ($2 \le x \le d-2$). We have:

$$D \le 2 * \frac{2^{d-1}}{2^{x+1}-2} + \frac{2^{x+1}-2}{2} \qquad (2.3)$$

The 2nd phase takes $(2^{x+1}-2)/2$ steps. This is because, if the remaining distance to the destination is larger than $(2^{x+1}-2)/2$, we may use the 3rd degree edge once more and then go back along the cycle.

Since we are considering very large number of vertices ($N=2^d$), to simplify the calculation, we rewrite the equation above as follows:

$$D \le \frac{2^d - 2^{d-x} + 2^{d-x}}{2^{x+1}-2} + 2^x - 1 = 2^{d-x-1} + \frac{2^{d-x}}{2^{x+1}-2} + 2^x - 1 \qquad (2.4)$$

To minimize the right side expression from (3.4), we take,

$$x = \frac{d-1}{2} \qquad (2.5)$$

thus:

$$D \leq 2^{\frac{d+1}{2}} \tag{2.6}$$

So, when the 3rd dimension of the partial Knödel graph is (d-1)/2, the maximum number of hops in our routing algorithm is of $O(\sqrt{N})$.

Now we consider the partial Knödel graph of higher degree. Suppose we use i degrees at each vertex ($1 \leq i \leq d$-3) with dimension 0 and 1. We denote the i dimensions to be used as $x_1, x_2, \ldots, x_i$ ($2 \leq x_1 < x_2 < \ldots < x_i \leq d$-2). Using edges of dimension 0 (or dimension 1, in the first step) and x_j, in every 2 hops, we can move closer to the destination by $2^{x_j+1} - 2$ edges along the Hamiltonian cycle. Thus the routing strategy in this graph is intuitive:

- Determine in which half the destination node is located.
- Determine the distance between the destination and current position along the cycle.
- Use the edges of the appropriate dimension and dimension 0 or 1 to approach the destination.
- Use dimension 0 and 1 alternatively to reach the destination along the Hamiltonian cycle.

With this routing strategy, the maximum number of hops is:

$$D \leq 2^{d-1-x_i} + 2^{x_i - x_{i-1}} + 2^{x_{i-1} - x_{i-2}} + \cdots + 2^{x_1} \tag{2.7}$$

To minimize the right side of (3.7)D, we should have:

$$d - 1 - x_i = x_i - x_{i-1} = \cdots = x_1 \tag{2.8}$$

Therefore,

$$\begin{cases} x_1 = (d-1)/(i+1) \\ x_2 = 2(d-1)/(i+1) \\ \dots \\ x_i = i(d-1)/(i+1) \end{cases} \tag{2.9}$$

and,

$$D \le (i+1) * 2^{(d-1)/(i+1)} = (i+1) * \sqrt[i+1]{N/2} \tag{2.10}$$

That is, when x_1, x_2, ..., x_i distribute evenly between 1 and d-1, the maximum number of hops of this routing strategy is minimized, which is of $O((i+1) * \sqrt[i]{N})$.

Routing in the Full Knödel Graph

The routing algorithm described above for the partial Knödel graph also applies to a full Knödel graph $W_{d,2^d}$. In the full Knödel graph with this strategy, we only utilize the dimensions 0 to d-2. The number of hops will be of $O(\log N * N^{1/\log N}) = O(\log N)$.

Although the shortest path problem is still an open problem for the Knödel graphs, some heuristics do exist, especially for $W_{d,2^d}$. First, we briefly introduce the heuristics, and then we use one of them to implement the routing algorithm in the P2P network based on the Knödel graphs.

With the alternative definition of Knödel graph $W_{d,2^d}$ illustrated in formula 3.2, each node of the graph is labelled by a d-bit integer, and there exists an edge between node x (x is even) and node $x+2^{k+1}-3$ *(mod N)*, for each $0 \le k \le d-1$.

Again, because Knödel graphs are vertex-symmetric, without loss of generality, we consider the routing starting from node 0. If there is a path between node 0 and node x using f edges of dimensions: t_1, t_2, …, t_f, it means that:

$$x = \sum_{i=1}^{f} (-1)^{i-1}(2^{t_i+1} - 3)(\bmod n) \qquad (2.11)$$

Since the Knödel graph is bipartite ([27]), *f* is even for even *x*. Therefore *x* can be represented as:

$$x = \sum_{t=1}^{m} (2^{i_t} - 2^{j_t}) \qquad (2.12)$$

where $d \geq i_1 > j_1 > i_2 > j_2 > \cdots > i_m > j_m > 0$. We can see that each $(2^{i_t} - 2^{j_t})$ pair in equation 3.12 actually represents a run of 1's in the binary representation of *x*. For example, when *d*=10, *x*=414, we have:

$$x = 414 = (0110011110)_2 = (2^{8+1} - 2^{6+1}) + (2^{4+1} - 2^{0+1}) \qquad (2.13)$$

Equation 3.13 actually gives a path from node 0 to node 414: dimension 8, dimension 6, dimension 4, and dimension 0 (node 0 – node 509 – node 384 – node 413 – node 414). The length of this path is 4.

Another property of the Knödel graph is that we can permute the dimensions in the path without affecting the routing length while the path will still lead to the same destination node. For example, if we re-write equation 3.13 as:

$$x = 414 = (0110011110)_2 = (2^{4+1} - 2^{6+1}) + (2^{8+1} - 2^{0+1}) \qquad (2.14)$$

The path consisting of edges of dimension 4, dimension 6, dimension 8, and dimension 0 is equivalent to the one above (node 0 – node 29 – node 928 – node 413 – node 414). The length of this path is still 4. It means that there exist several different routes of the same length between the same pair of nodes. This property makes the network based on the Knödel graph more robust compared with the CHORD network, where the number of the paths is always 1 according to the protocol.

Using the routing algorithm in CHORD, the routing length between the same two nodes is 6 (node 0 – node 256 – node 384 – node 400 – node 408 – node 412 – node 414).

It turns out that the binary representation of even number x describes a path between node 0 and node x. Even in the worst case, the length will be d, which is of $O(\log N)$ in the worst case. With the reduction rules introduced in [30], this length may be reduced down to $\lceil \frac{d+2}{2} \rceil$.

In order to introduce the reduction rules, let's first make some definitions.

Recall the equation (3.12), $x = \sum_{t=1}^{m} (2^{i_t} - 2^{j_t})$, let

$I = \{i_1, i_2, \cdots, i_m\}$, $J = \{j_1, j_2, \cdots, j_m\}$

- Reduction rule R1

if $a \neq b \in I$ such that $a-1, b+1 \in J$ exist, then

do

$$I = I \cup \{a-1\} - \{a,b\}$$
$$J = J \cup \{b\} - \{a-1, b+1\}$$

done

- Reduction rule R2

if $a,b,c \in I$ are pair-wise not equal such that

$a-1, b-1, c+2 \in J, a-1 \neq c+2 \neq b-1$ exist and $c+1 \notin I \cup J$, then

do

$$I = I \cup \{a-1, b-1\} - \{a,b,c\}$$
$$J = J \cup \{c, c+1\} - \{a-1, b-1, c+2\}$$

done

- Reduction rule R3

if $a,b,c \in I$ are pair-wise not equal such that

$a+1, b+1, c-2 \in J, a+1 \neq c-2 \neq b+1$ exist and $c-1 \notin I \cup J$, then

do

$$I = I \cup \{c-1, c-2\} - \{a,b,c\}$$
$$J = J \cup \{a,b\} - \{a+1, b+1, c-2\}$$

done

- Reduction rule R4

if $d,a \in I, d \neq a$, $a-1 \in J$, then

do

$$I = I \cup \{a-1\} - \{a,d\}$$
$$J = J - \{a-1\}$$

done

The correctness of the rules follows from the equivalences:

R1: $2^a + 2^b - 2^{a-1} - 2^{b+1} = 2^{a-1} - 2^b$

R2: $2^a + 2^b + 2^c - 2^{a-1} - 2^{b-1} - 2^{c+2} = 2^{a-1} + 2^{b-1} - 2^{c+1} - 2^c$

R3: $2^a + 2^b + 2^c - 2^{a+1} - 2^{b+1} - 2^{c-2} = 2^{c-1} + 2^{c-2} - 2^a - 2^b$

R4: $2^d + 2^a - 2^{a-1} (\mathrm{mod}\, 2^d) = 2^{a-1}$

We can also have some more complicated rules, such as R5:

$$2^a + 2^b + 2^c + 2^e - 2^{a-1} - 2^{b-1} - 2^{c-1} - 2^{e+3} = 2^{a-1} + 2^{b-1} + 2^{c-1} - 2^{e+2} - 2^{e+1} - 2^e$$

or R6:

$$2^a + 2^b + 2^c + 2^e - 2^{a+1} - 2^{b+1} - 2^{c+1} - 2^{e-3} = 2^{e-1} + 2^{e-2} + 2^{e-3} - 2^a - 2^b - 2^c$$

and so on. Practice shows that when the degree of the selected Knödel graph is 32 or less, the number of hops reduced by the complicated rules is limited. Considering the fact that these rules need much more computing time, we will apply only the 4 rules $R1$-$R4$ listed above in our application.

Fertin et al ([30]) proved that by processing the representation in formula 3.12 with the rules $R1$, $R2$, $R3$ and $R4$ until none of them applies any more, we will have:

$$\#(I \cup J) \leq \left\lceil \frac{d+2}{2} \right\rceil \qquad (2.15)$$

When x is odd, according to the 3 least significant bits of x, by re-writing the representation of x in the following format:

$$x = \sum_{t=1}^{m} (2^{i_t} - 2^{j_t}) + 2^s - 3, (s \in \{1,2,3\}) \qquad (2.16)$$

(3.15) still holds (more details in [30]).

Practically, analysing all the different cases of reducing $\#(I \cup J)$ to the upper bound stated in formula 3.15 turns out to be too complicated and requires too much computing in the real P2P network. Our experiment shows that, by properly selecting the order to apply the 4 rules in a simple way, we can reduce the path length to $d/3$ in average and $2d/3$ in maximum (see Chapter 4).

3.1.2 P2P Algorithms based on the Knödel Graph

As already discussed, the fundamental problem of a peer-to-peer system is to efficiently locate the peer that stores a desired data item. With the basic routing strategy, now we can describe our peer-to-peer protocol based on the partial Knödel graph, which addresses the content locating problem.

Basic design

CHORD designed some good algorithms and protocols to maintain the P2P network. Our P2P architecture has similarities with CHORD in some aspects. In terms of the following items, we will follow or reuse some algorithms from CHORD with slight modification:

1. We associate each data item with a key, and hash the keys to each node of the Knödel graph. The keys might be the names of the files to be shared in the system, or might be the keywords of the shared content, depending on different applications. The peer holding a key stores the address of the peer that owns the content associating with the key.
2. Considering all the nodes of the Knödel graph as a space, the entire space is dynamically partitioned among all the peers in the system, and each peer is actually in charge of an arc of the Hamiltonian cycle of the Knödel graph.
3. Each peer maintains a routing table which contains d items (or less, if using the partial Knödel graph). Different from CHORD, here the routing table is constructed according to edges in the Knödel graph.
4. When searching for certain content, first find the node holding the key, and then find the peer in charge of the node, forward the querying message to the peer, and retrieve the address of the peer which owns the content.

Here we use the numbering strategy described by formula (3.2) for the Knödel graph. This means that the number space of the graph contains integers in the range of [0, 2^d) for $W_{d,2^d}$.

The main differences between the MP2P infrastructures based on the Knödel graph and that of CHORD are:

1. The routing strategies are different. The average path length in our P2P network is approximately half of that in CHORD.
2. The Knödel graphs are bi-directional (CHORD is not), i.e., if node a has an entry in its routing table connecting to node b with an edge of dimension k, then node b also has an entry in its routing table connecting to node a with the same dimension k. This makes some algorithms much simpler. For example, when a peer leaves the network, it may inform the peers with connection to it, so that those peers do not need the stabilization to correct their routing table.
3. There exist several different paths for each routing. This makes the MP2P network based on the Knödel graph more robust than CHORD.

In the following discussion, we consider the MP2P infrastructure based on the full Knödel graph $W_{d,2^d}$ if not specified otherwise. For the infrastructures based on the partial Knödel graph, they share mostly the same properties and algorithms. The only difference is that with the partial Knödel graph, each peer maintains fewer links to other peers, and the algorithms will use only these links to do routing and maintaining. The value of d is configurable depending on the application. It should make 2^d large enough to avoid frequent duplicated hashing values. In the simulation that will be introduced in chapter 4, we choose $d = 31$.

Routing

Each peer maintains a routing table of size *d*. In real applications, the contents of the routing table may be the IP addresses or URLs of the other peers. In our simulation, we use pointers pointing to other peer objects. For the peer with the even key value *k*, the *i*-th ($1 \leq i \leq d$) item in the routing table points to the peer with key value k', where $k' \geq (k + 2^i - 3) \bmod n$, and there does not exist another peer *p* with key value k'' where $k + 2^i - 3 \leq k'' \leq k'$.

Besides the routing table, each peer also keeps the information of its nearest predecessor (if peer *p* is peer *q*'s successor, then *q* is *p*'s predecessor). The successor's information is not needed explicitly because it will show up in the second row of the routing table. With the routing tables on the peers, we derived the routing algorithm for the P2P network from the routing algorithm in the partial Knödel graph introduced in 3.1.1. The algorithm is presented below:

```
peer::findKey (key)
{
        if( current peer is in charge of key )
                return this;
        peer1 = closest successor to key in routing table;
        if( peer1 = peer )
                peer1 = predecessor;
        return peer1.findKey(key);
}
```

In the algorithm, line 6 and 7 are needed because it is possible that there is (are) peer(s) located before the current peer but after the destination key.

Practically, considering the procedure of recovering and maintenance in case of peer failing, line 5 can be rewritten as:

```
5:          peer1 = closest available successor to key in routing table;
```

Furthermore, considering the possibility that the predecessor may fail, line 7 should be modified as:

```
7a:                 peer1 = predecessor;
7b:                 peer1 = the furthest peer available in routing table
```

Here the distance between two peers is the distance on the Hamiltonian cycle. A simple method to test the availability of a peer in the TCP/IP (mobile IP may be involved) network is to send the ICMP echo request packet to the peer (or use the "ping" command, if available on the device).

Based on the observation that the edges of the same dimension point to the different direction on an even numbered node and an odd numbered node, in our simulation, we also test the case that each peer holds two sets of routing table, one for even node $2n$, and another for odd node $2n+1$. The simulation results show that this change makes the performance more consistent. To avoid using larger sized routing table, we can always put the peers on the even numbered nodes. This setting is totally optional. Applications may choose this configuration according to the computing ability of the peers.

Construction

The basic steps of peer joining of our P2P network are similar to those of CHORD:

1. Initializing the new peer's routing table and predecessor;
2. Updating the routing table of existing peers;
3. Transferring keys to the new peer from its successor.

However, because of the different routing mechanism, details of step 1 and 2 are different from those in CHORD. The first thing the bootstrapping peer needs to do is to check if it has to update its own routing table. This can be done by scanning each dimension in the routing table to see if any of them should be replaced by the new peer. The remaining of step 1 is as follows:

```
1:  peer::initializeRoutingTable(newPeer)
2:  {
3:          update this->routingTable;
4:          for each item of newPeer's routing table
5:          do
6:                  suc = findKey(newPeer->key + offset of the item);
7:                  set the item of newPeer's routing table as suc;
8:          done
9:  }
```

The "offset of the item" in the above algorithm is related to the dimension of the link in the Knödel graph. For link of dimension k, the offset is $(2^{k+1}-3) \bmod 2^d$. The updating of the predecessor and the routing tables of existing peers rely on the thread that every peer runs periodically to verify the correctness of its routing table. This thread also helps to fix the routing table in case of peer failure. The algorithm of the thread is shown below:

```
peer::verifyRoutingTable( )
{
        //check predecessor
        if( predecessor has failed ) {
                predecessor = null;
        }
        //validate routing table
        from the highest to the lowest dimension in the routing table {
                routing table item = findKey(key+offset);
        }
        //notify successor
        successor.notify(this);
}
peer::notify( peer1 )
{
        //peer1 may be the new predecessor
        if( predecessor == null || predecessor.key < peer1.key ) {
                predecessor = peer1;
        }
}
```

Peer departure, recovery and maintenance

Basically when a peer departs or fails, the algorithm of verifying routing table and predecessor introduced above may fix the routing tables of the affected peers. If the configuration setting makes each peer to hold 2 sets of routing table (for node $2n$ and $2n+1$), peers that plan to leave may also notify the nodes in its routing table. In this way the affected peers may fix their routing table faster without waiting for the timeout of the fixing thread. Below is the leaving notification algorithm for this case:

```
1:  peer::leavingNotification( )
2:  {
3:          transfer keys to predecessor
4:          notify predecessor to update its successor
5:          from each node in the routing table {
6:                  send predecessor and successor info to the node
7:          }
8:  }
```

Another algorithm we propose for the MP2P network is that, each peer with content to share periodically runs the thread to check if the peer that should be holding the key of its contents is actually holding it. If not, it will send the notice to the later peer to add the content to its list. This helps in case of unexpected peer failure. Below is the algorithm:

```
peer::checkContents(keyOfContent)
{
        peer1 = findKey(keyOfContent);
        if peer1 does not have the content in its list {
                peer1->addContent(keyOfContent, this);
        }
}
```

3.1.3 Efficiency Analysis of the P2P Algorithms on Knödel Graph

Number of peers need to be contacted during querying

The routing length in the Knödel graph $W_{d,2^d}$ is of $O(\log N)$, while in the partial Knödel graph, it is of $O(\sqrt[i+1]{N})$ (where i is the degree of the graph, N is the total number of nodes). In the P2P architecture based on the Knödel graph (or partial Knödel graph), the routing length depends on the hashing algorithm we employ to distribute the peers on the node space. When the hashing algorithm distributes the peers evenly on the node space, with high probability, the proposition that the routing length is of $O(\log N)$ for the full Knödel graph and $O(\sqrt[i+1]{N})$ for the partial Knödel graph still holds (here N is the total number of peers that have joined the MP2P network). There are simulation results in Chapter 4 that verify this statement.

Number of keys each peer maintains

The number of keys each peer maintains relies on the hashing algorithm. It has nothing to do with the routing. Therefore, the MP2P network based on the Knödel graph has the same property as that of CHORD from this point of view. According to [10], in the network with N peers and K keys, each peer maintains at most $(1+\varepsilon)K/N$ keys.

Size of routing table

For the full Knödel graph $W_{d,2^d}$, the size of the routing table is d. For the partial Knödel graph, the size is i (where i is the degree of the graph). In practice, some dimensions will be pointing to the same peer. For example, in our simulation, where $d = 31$ and the number of peers is between 1000-10000, dimensions 1-18 usually pointing to the same peer (which is the successor). By slightly changing the data structure of the routing table (using the range of the dimensions instead the exact dimension), we can reduce the size of the routing table by at least half.

Query failure

Unfortunately routing failure still happens in some special cases. For example, peer p's predecessor is $p1$, and $p1$ holds the key k; if $p1$ fails, before the owner of the content with key k find out that $p1$ failed and notifies p to add key k to its list, querying of key k will fail. By running the thread *peer::checkContents*, this failure may be solved by retrying the query after a configurable period.

3.1.4 Improvements in Mobile Network

Considering the unstable nature of the mobile peers, a mobile device does not necessarily join the MP2P structure directly. Instead, it may link to the network via a peer which is more stable in the MP2P architecture. This makes the whole network a 2-leveled structure: the Knödel graph (or partial Knödel graph) on the top and some other underneath nodes linking to the peers on the top level. Figure 4 shows an example of this structure.

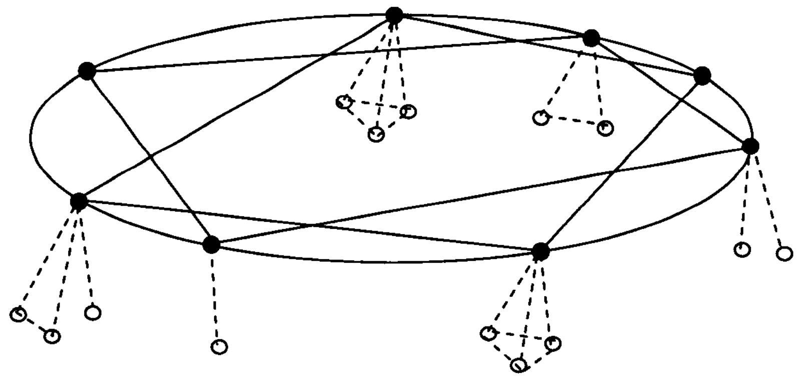

Figure 4 – The 2-leveled MP2P Structure

This structure is somehow similar to that of the Grapes introduced in [25]. The difference is that Grapes is based on CAN ([11]). Basically, in the 2-leveled structure, the routing length is about half of that in the flat structure.

The idea of putting peers in 2 different levels seems to be against the concept of "each peer is equally responsible in the P2P network" (comparing with the client/server structure). However, differences do exist between peers. Some peers have better computing ability, some others have wider bandwidth. A powerful peer may usually become the proxy or router of several other peers. In mobile communication, there exist many forms of this kind of small community. It could be a wireless LAN on a vehicle, a Bluetooth PICO, and it also could be several devices without any direct relationship but just locating nearby. In a WAN project, several mobile devices, such as PDAs, laptops and tablets, connect to the public network through a mobile router which supports the mobile IP. If we form the network with the 2-leveled structure, the performance and stability will be greatly improved compared with the flat structure. The mobile routers in the network may act as the leaders of each LAN behind it, and may cache certain contents thus save quite a lot of traffic.

4 Simulation Results

There are two parts of our simulation. Section 4.1 shows the results of algorithms used to reduce path length in the Knödel graph, and section 4.2 shows the simulation results of the MP2P network performance.

The simulation results are compared with that of CHORD and CAN.

4.1 Routing in the Knödel graph

4.1.1 Routing in a full Knödel graph

Our experiment was done in the Knödel graph $W_{31,2^{31}}$. We implemented the reduction rule R1-R6 introduced in 3.1.1. We randomly selected 4 groups of destination nodes, with 1000 nodes in each group, and calculated the average path length from node 0 to these nodes, both before the reduction and after the reduction. The purpose of dividing the testing nodes into four groups is to verify if the performance of the algorithm is consistent. The statistic values of how each rule reduced the length are also counted and displayed in the table.

The following table shows the results of our first experiment, where we simply applied the rules with order R4–R2–R3–R1 to all the destination nodes.

Table 4.1 Routing length in the Knödel graph

	Group 1	Group 2	Group 3	Group 4
Average routing length before reduction	15.5	15.4	15.5	15.5
Average routing length after reduction	11.5	11.3	11.5	11.5
Maximum routing length before reduction	24	24	26	24
Maximum routing length after reduction	18	16	18	18
Total hops eliminated by rule R1	303	832	366	487
Total hops eliminated by rule R2	299	795	415	509
Total hops eliminated by rule R3	282	796	404	503
Total hops eliminated by rule R4	308	819	373	488

Table 4.2 shows the result after changing the order of the rules to R1–R2–R3–R4.

Table 4.2 Routing length in the Knödel graph – another order of reduction

	Group 1	Group 2	Group 3	Group 4
Average routing length before reduction	15.5	15.4	15.5	15.5
Average routing length after reduction	11.5	11.3	11.5	11.5
Maximum routing length before reduction	24	24	26	24
Maximum routing length after reduction	18	16	18	18
Total hops eliminated by rule R1	1828	1809	1802	1789
Total hops eliminated by rule R2	126	121	100	117
Total hops eliminated by rule R3	49	58	73	62
Total hops eliminated by rule R4	0	0	0	0

We can see from the table above that the average and maximum routing length keeps almost the same value. The difference is the number of hops eliminated by each reduction rule.

These two tables show that, for the Knödel graph $W_{d,2^d}$, simply applying the reduction rules in the same order may reduce the average routing length by 1/3 (*d*/2 before reduction, *d*/3 after reduction), and the maximum routing length is about 2*d*/3.

By implementing the algorithm in a much more complicated way, i.e., carefully choosing the order of the rules for different nodes, we may approach the theoretical upper bound for the maximum routing length, $\lceil (d+2)/2 \rceil$.

In the base network of CHORD of the same size, the degree of the graph is 31, the average and maximum routing lengths in our case are 15.5 and 31 respectively.

4.1.2 Routing in a partial Knödel graph

The following table shows the results of routing length in the partial Knödel graphs of degree 10, 15 and 20. The routing algorithm is shown in (3.1.1). The selected destination nodes in this experiment are the same as those in 4.1.1.

Table 4.3 shows that, with 1-3 less degree than the CHORD graph, the average routing length is the same as or very close to that in CHORD.

Table 4.3 Routing length in the partial Knödel graph

	Group 1	Group 2	Group 3	Group 4
Average routing length when degree=10	44	44	44	43
Maximum routing length when degree=10	74	70	73	72
Average routing length when degree=15	25.3	25.4	25.3	25.2
Maximum routing length when degree=15	41	41	40	38
Average routing length when degree=20	21.8	21.6	21.8	21.6
Maximum routing length when degree=20	35	33	36	35
Average routing length when degree=25	18.5	18.6	18.5	18.4
Maximum routing length when degree=25	30	30	28	31
Average routing length when degree=28	16.8	16.7	16.9	16.8
Maximum routing length when degree=28	27	26	27	26
Average routing length when degree=29	16.2	16.3	16.3	16.2
Maximum routing length when degree=29	25	25	27	25
Average routing length when degree=30	15.7	15.7	15.9	15.7
Maximum routing length when degree=30	25	25	26	24

4.2 Simulation of the P2P Network

We concentrated on the routing algorithm of the P2P network based on the Knödel graph. Again, the base graph is $W_{d,2^d}$, where d = 31. When doing the routing experiment, there are approximately 4096 peers in the network. In the CHORD network of the same size, the average routing length in theory is 6. In practice, it is around 7-8 because of the random distribution of the peers on the space. In CAN, if the number of space dimension is 15, the average routing length is 6.5 in theory.

In our experiment, we randomly chose some keys, and calculated the average routing length for all the peers to query the key. The following table shows the results of part of the data and the total average.

Table 4.4 Routing Length in the MP2P Network

Key	Average Routing Length	Max Routing Length
0x00002a11	5.53	10
0x1234ac50	5.13	10
0x023583ab	5.65	10
0x0004ab22	4.78	8
0x000001ef	5.53	10
0x2311efaa	5.33	10
0x521d34e2	4.49	8
0x62aa56a1	5.76	10
0x722aa687	4.12	8
0x32cab6e8	4.72	10
Total Average	5.10	9

Table 4.5 below shows the size of the routing table of the peers. In the CHORD network, the average size of the routing table is 31. In CAN, the average size of routing table is 30 when the number of space dimension is 15.

Table 4.5 Size of Routing Table in the MP2P Network

Average size	Maximum size	Minimum size
14.3	18	11

5 Conclusions and Future Work

The (full) Knödel graphs are well known to have good communication properties. In this thesis, we proposed a new P2P topology based the Knödel graphs, and we implemented the routing algorithm in the Knödel graph and partial Knödel graph. The simulation results show that, without significantly increasing the computing complexity or computing time, the average routing length of the P2P network based on the Knödel graph is approximately 2/3 of that in CHORD (the same ratio for the maximum routing length). In the mobile networks, this may save a lot of air traffic which is expensive. Shorter routing length also means that the network is more robust, because a peer contacts fewer peers when querying for certain contents. In case of the node failure, the recovering process will affect fewer peers.

From the binary representation, the routing path of length less or equal to d (or, log$N$) can be easily found for Knödel graph $W_{d,2^d}$. By implementing the reduction rules, this maximum length can be reduced to $\lceil (d+2)/2 \rceil$.

The routing length in the partial Knödel graph is of $O(\sqrt[i+1]{N})$, where i is the degree of every node. So, our P2P network can have some applications, especially in the mobile environment where the computing resources are limited on some mobile devices.

The shortest path problem of the Knödel graph is still an open problem. The reduction rules introduced in (3.1.1) only set an upper bound of the maximum length. The method of finding the shortest path between any two general nodes in the graph remains to be a challenging question.

Future work may also involve implementing the reduction rules to the MP2P network based on the Knödel graph, so that the routing length can be reduced further.

As mentioned above, the robustness of the MP2P network based on the Knödel graph is improved comparing with that of CHORD. However, a malicious or buggy peer in the network could still cause misshape of the whole network.

Bibliography

[1] Napster website, 15 October 2005 <http://www.napster.com>

[2] Gnutella website, 15 October 2005 <http://www.gnutella.com>

[3] Freenet website, *The Freenet Protocol*, 10 November 2005 <http://www.freenetproject.org>

[4] KaZaA website, 10 November 2005 <http://www.kazaa.com>

[5] Jabber website, 18 November 2005 <http://www.jabber.com>

[6] SFLan website, 18 March 2007 <http://www.archive.org/web/sflan.php>

[7] Groove website, 18 November 2005 <http://www.groove.net>

[8] SETI@Home website, 20 November 2005 <http://setiathome.ssl.berkeley.edu>

[9] T. Leighton. *Introcution to Parallel Algorithms and Architectures: Array-Trees-Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, California, 1992.

[10] I.Stoica, R.Morris, D.Karger, F.Kaashoek, H.Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001. pp.149-160

[11] S.Ratnasamy, P.Francis, M.Handley, R.Karp, S.Shenker. A Scalable Content-Addressable Network, *Proc. ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001. pp.161-172.

[12] B. Yang and H. Garcia-Molina, Improving Search in Peer-to-Peer Networks, *Proceedings of the 22nd International Conference on Distributed Computing Systems* (ICDCS'02). 2002. pp.5-14.

[13] H.Johansen and D.Johansen, Improving Object Search Using Hints, Gossip, and Supernodes, *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Oct.13-16, 2002, Osaka University, Suita, Japan. pp.336-340.

[14] K.Aberer, M.Punceva, M.Hauswirth and R.Schmidt, Improving Data Access in P2P Systems, *IEEE Internet Computing*, Jan.2002. pp.58-67.

[15] A.Andrzejak and Z.Xu, Scalable, Efficient Range Queries for Grid Information Services, *2nd IEEE International Conference on Peer-to-Peer Computing*, Sep.5-7, 2002, Linköping, Sweden. pp.33-40.

[16] A.Kothari, D.Agrawal, A.Gupta and S.Suri, Range Addressable Network: A P2P Cache Architecture for Data Ranges, *Proceedings of the 3rd International Conference on Peer-to-Peer Computing* (P2P'03). 2003. pp.14-22.

[17] M.Ripeanu and I.Foster, Mapping the Gnutella Network, *IEEE Internet Computing*, Jan.2002, pp.50-57.

[18] M.K.Ramanathan, V.Kalogeraki and J.Pruyne, Finding Good Peers in Peer-to-Peer Networks, *Proceedings of the International Parallel and Distributed Symposium*, IPDPS'02. 2002. pp.24-31.

[19] L.Zou, E.W.Zegura and M.H.Ammar, The Effect of Peer Selection and Buffering Strategies on the Performance of Peer-to-Peer File Sharing Systems, *Proceedings of the 10th IEEE Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, MASCOTS'02. 2002. pp.63-70.

[20] B.Leuf, *Peer to Peer: Collaboration and Sharing over the Internet*, Addison-Wesley Person Education, June 2002.

[21] R.Grimm and J.Nützel, Peer-to-Peer Music-Sharing with Profit but Without Copy Protection, *Proceedings of the Second International Conference on WEB Delivering of Music* (WEDELMUSIC'02). 2002. pp.17-22

[22] H.A.Harutyunyan and A.L.Liestman. Messy Broadcasting, *Parallel Processing letters*, vol. 8, no. 2, 1998, p149-159.

[23] H.A.Harutyunyan, Multiple message broadcasting in modified Knödel graph, *7th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2000), Proc. in Informatics*, Carleton Scientific, 2000.

[24] D.Karger, E.Lehman, F.Leighton, M.Levine, D.Lewin and R.Panigrahy, Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, May 1997, pp.654-663.

[25] K.Shin, S.Lee, G.Lim, H.Yoon and J.S.Ma, Grapes: Topology-based Hierarchical Virtual Network for Peer-to-peer Lookup Services, *Proceedings of the International Conference on Parallel Workshops*, 2002 (ICPPW'02). pp.159-164.

[26] W. Knödel. New gossips and telephones. *Discrete Mathematics*, 13, 1975. pp.95

[27] G.Fertin and A.Raspaud, A Survey on Knödel Graphs, *Discrete Applied Mathematics*, Vol 137, Issue 2 (Mar. 2004). pp.173-195.

[28] P. Fraigniaud and J.G. Peters, Minimum linear gossip graphs and maximum linear (Δ, k)-gossip graphs. *Networks*, Volume 38, Number 3, October 2001. pp.150-162.

[29] C.D.Morosan, New Communication Properties of Knödel Graphs, *Master Thesis*, Concordia University, 2004.

[30] G. Fertin, A. Raspaud, O. Sýkora, H. Schröder, and I. Vrto, Diameter of Knödel Graph, *26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG2000)*, Vol 1928 of Lecture Notes in Computer Science, Springer-Verlag, 2000. pp.149-160.

[31] J.-C. Bermond, H.A. Harutyunyan, A.L. Liestman, and S. Perennes. A note on the dimensionality of modified Knödel graphs. *IJFCS: International Journal of Foundations of Computer Science*, 8(2):109-116, 1997.

[32] C. Perkins, IP Mobility Support for IPv4, 23 March 2006 <http://www.ietf.org/rfc/rfc3344.txt?number=3344>

[33] T.Hossfeld, K.Tutschku, F.U.Andersen, Mapping of file-sharing onto mobile environments: feasibility and performance of eDonkey with GPRS, *Wireless Communications and Networking Conference*, 2005 IEEE Volume 4, 13-17 March 2005, pp.2453 – 2458.

[34] T.Hossfeld, K.Tutschku, F.U.Andersen, Mapping of file-sharing onto mobile environments: enhancement by UMTS, *PerCom 2005 Workshops. Third IEEE International Conference on Pervasive Computing and Communications Workshops*, 2005. pp.43-49.

[35] eDonkey website, 12 October 2005 <http://www.edonkey.com> (the site is not available as of 10 April 2007)

[36] H.Harutyunyan, J.He A New Peer-to-peer Network. *PerCom 2007 Workshops. Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007. pp.120-125.