HiPeer: An Evolutionary Approach to P2P Systems

vorgelegt von
Diplom-Ingenieur
Giscard Wepiwé

Von der Fakultät IV – Elektrotechnik und Informatik –
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzende:	Prof. Dr. Sabine Glesner
Berichter:	Prof. Dr.-Ing. habil. Sahin Albayrak
Berichter:	Prof. Dr.-Ing. Erik Mähle

Tag der wissenschaftlichen Aussprache: 19. Juni 2006

Berlin 2006

D 83

To my beloved wife Daniele and

Milina my dear daughter,

both are gift of god.

Contents

List of Tables

Listings

List of Figures

Abstract

The current work deals with the problem of resource management in P2P systems. We have proposed an evolutionary concept that enables self-management of nodes and resources in P2P networks. The approach is evolutionary in the sense that it unveals a new step in the P2P research. The work achieved in this dissertation can be subdivided in two main parts:

In the first part, we have developed a structured approach of a P2P architecture that organizes nodes and resources of a P2P network in a concentric multi-ring (CMR) overlay. Resources are replicated by using a slightly modified approach of the distributed hash table (DHT) concept. Each data item is remotely located on a set of nodes sharing the same prefix with the key associated to that data item. This can be exploited for supporting range queries while preserving load balancing in the network. The routing on the CMR structure is based on the graph theoretical concept of De Bruijn, and each node maintains a maximum limited number of $2\Delta + 2$ entries in its routing table, independently of the number $N = \frac{\Delta^{D_{max}+1}-\Delta}{\Delta-1}$ of the nodes in the system. The parameter $\Delta \geq 2$ denotes the degree[1] of the De Bruijn digraph formed by each ring in the CMR overlay.

In the second part, we have proposed a churn-resistant strategy for managing the dynamic behavior of the nodes in a CMR based network. We showed that if each node in the network periodically retransmits no more than one KEEPALIVE message (heartbeat message) to one of its neighbors in the network, any node's failure can be detected and recovered so that the network is kept in a consistent state. Additionally, we demonstrated that even if half of the nodes forming the P2P network fail collectively, the failures can be detected and recovered so that

[1]The degree of a vertex $\Delta(x)$ is the number of vertices adjacent to x. The degree of a graph G is $\Delta_G = \max\{\Delta(x), x \in V\}$. D_{max} is the total number of rings.

a consistent state of the network is re-established.

In terms of performance, it has been proved that for any network with at least $N = \Delta$ nodes, any existing object or resource can be located within at most $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ hops with a probability of over 99%. The lookup value D_{CMR} is so far the lowest lookup bound comparing to other outstanding resource location strategies in P2P systems with the same number of entries in the routing table. We showed that the management load of the network is fairly balanced between the nodes in the system and that the management cost for node's insertion or deletion is relatively low. In fact, a node joining or leaving the network involves at most $2\Delta + 2$ other nodes in the network management process.

Zusammenfassung

Diese Dissertation behandelt das Problem der Ressource Verwaltung in Peer-to-Peer (P2P) Systemen. Als Lösung wurde ein evolutionäres Konzept für die Selbst-Organisation der Knoten in einem P2P Netzwerk vorgeschlagen. Das Konzept trägt den Namen "HiPeer" und wurde in zwei Teilen beschrieben:

HiPeer besteht in erster Linie aus einem strukturierten P2P Netzkonzept, wobei Knoten in einer konzentrische Multi-ring (Concentric Multi-Ring: CMR) Architektur repräsentiert werden. Die Einbringung von neuen Ressourcen wird nach einer leicht veränderten Version der verteilten hashing Tabellen (Distributed Hash Tables: DHT) realisiert, wo jedes Ressource-Element in einer Liste von Knoten mit gleichen Präfixen (Präfixen der Identifikationsnummer) repliziert wird. Das Routen zwischen den Knoten in der CMR-Architektur basiert auf dem altbekannten Graphkonzept von De Bruijn. Jeder Knoten hat eine Routingstabelle mit einem Maximum von $2\Delta + 2$ Einträgen, unabhängig von Anzahl der Knoten $N = \frac{\Delta^{D_{max}+1}-\Delta}{\Delta-1}$ im System. Der Parameter $\Delta \geq 2$ ist der Grad [2] des De Bruijn Digraphs, der von jedem Ring konstruiert wird.

Im zweiten Teil basiert HiPeer auf einer dynamik-resistenten Strategie, um die Probleme der ständigen Veränderungen in der Netztopologie zu bewältigen. Dabei wird nur eine einzige periodische Nachricht (eine KEEPALIVE Message) pro Knoten für das Aufrechterhalten der Netzkonsistenz ausgetauscht. Mit einer regelmässigen KEEPALIVE Nachricht pro Knoten kann man das System in einen konsistenten Zustand zurückführen, auch wenn bis zur Hälfte der Knoten im System gleichzeitig ausfallen.

Zusätzlich wurde in dieser Dissertation die Performanz des vorgeschlagenen P2P Ansatzes theoretisch analysiert und eine experimentelle Evaluierung durch

[2] Der Grad eines Knoten $\Delta(x)$ ist die Anzahl der benachbarten Knoten von x. Der Grad eines Graphs G ist $\Delta_G = \max\{\Delta(x), x \in V\}$. D_{max} ist die Anzahl der Ringe.

Simulation realisiert. Damit konnte man zeigen, dass jede Ressource im Netz mit einer Wahrscheinlichkeit vom über 99% Prozent und mit einem Suchweg der Länge unter $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ (Overlay Hops) gefunden werden kann. Dies ist möglich unter der Voraussetzung, dass das Netz aus mindestens $N = \Delta$ Knoten besteht. Der Suchweg der Länge D_{CMR} ist einer der Besten im Vergleich zu anderen sehr guten Lösungsvorschlägen, die einen Suchweg der Länge $\log N$ aufweisen. Man hat mit der vorgeschlagenen Lösung auch zeigen können, dass die Last zwischen den Knoten fair verteilt ist und der Managementaufwand für die Integration von neuen Knoten sowie die Restrukturierungskosten nach dem Ausfall von Knoten gering sind. Für die Verwaltung beim Einfügen oder Ausfall von Knoten im System sind höchstens $2\Delta + 2$ anderer Knoten im Stabilisierungsprozess involviert.

Acknowledgements

This thesis is the result of many years of effort and would not have been possible without the assistance of many great people. Here, I would like to thank all those who have assisted me to complete this work.

First, I would like to express my gratitude to my adviser Sahin Albayrak, whose steady guidance allowed me to achieve this thesis. He trusted me and gave me the chance to work in his laboratory, where I gained many experiences working in close cooperation with partners from industries and research institutions at national and international levels. His assistance has made this thesis possible, and I truly believe that his influence has helped me become a better researcher.

I would like to thank my second adviser Erik Maehle, who trusted me and accepted to review this thesis. He payed a special attention to my research topic as we met last summer in Boston at the Network Computing and Applications conference and we had a very interesting discussion. His comments and advices have been very helpful to improve the quality of this dissertation.

I would also like to express my sincere gratitude to Plamen who has assisted me during the last phase of this work. We had many hours of very interesting and fruitful discussions. His assistance was decisive for this thesis.

A special thank goes to my family, particularly to my parents. Thank you mum and dad for all the efforts and the energies you spent to ensure a good education for your children. You always believed in me and encouraged me. Your unconditional supports have made this work possible. *"Merci Papa, Merci Maman."*

Further, I would like to thank Steffen from the mathematical department who takes the time to carefully checking and proof reading the mathematical statements in this document. A special thank also goes to all my colleagues at the

DAI-Laboratory, my colleagues in other institutes at the Technische Universitaet Berlin. I would also like to thank all my friends for their encouraging words, when I seem to be confused.

Last but not least, I would like to express my extreme gratitude to my wife *Daniele* and my daughter *Milina* to whom I dedicate this work. Both, you were those who directly felt the pressure and the stress of my everyday life in the last phase of this thesis. This work could not have been achieved without your patience, your assistance and your love.

Giscard Wepiwé.

Chapter 1

Introduction

THe focus of this thesis is to research a novel and trustworthy solution for the problem of resource management in peer-to-peer (P2P) systems. P2P systems, — also referred as P2P networks or simply P2Ps in this document, — are computer networks or systems in which nodes or peers of equal roles and responsibilities, often with various capabilities, exchange information or share resources directly with each other. P2P systems can function without any central administration and coordination instance. Depending on the performing operation, a peer can simultaneously play the role of client, server and router as opposed to the traditional client-server paradigm, where nodes are either client, server or router. Besides, nodes in P2P systems are autonomous in the sense that: (i) they can join the system anytime, (ii) they can leave without any prior warning, and (iii) they can take routing decision locally in an ad-hoc manner. These fundamental characteristics of P2Ps enable the fast and cost-efficient deployment of self-managed computer systems with high overall management cost, but with low management cost at each peer. With P2Ps, one can deploy large-scale computer systems without the need of cost-intensive supercomputing infrastructure in which management is highly complex and requires high-skilled administrators

for their maintenance.

On the other hand, because of its main design principle of being completely decentralized and self-organized, the P2P concept paves the way for new type of applications such as file-sharing applications and collaboration tools over the Internet that has recently attracted tremendous user interest. Another application domain of interest is the sharing and aggregation of large-scale geographically distributed processing and storage capacities of idle computers around the globe to form a virtual super-computer as the Seti@Home project did [11]. P2P networking is an important enabling technology for the realization of self-managed and autonomous systems, where each node manages its own activities by itself, thus ensuring a consistent state of the system.

However, owing to the properties of their nodes which can join and leave continually, P2P systems are dynamic systems with high rate of churns and unpredictable topology. A direct consequence is that resources or nodes are restricted to temporary availability only. A network element can disappear at a given time from the network and reappear at another locality of the network with an unpredictable pattern. Under these circumstances, one of the most challenging problems of P2Ps is to manage the dynamic and distributed network so that resources can always successfully be located by their requesters when needed. Our intention in this thesis is to develop a resource management solution for P2P systems that takes into account the main characteristics of such networks. What is needed is a resource management strategy that is suitable for practical P2P applications in large-scale environments and one that can support real-time or nearly real-time applications like distributed online gaming and distributed resource sharing for power-intensive applications. In the sequel, we describe the main lines of the resource management problem with the limitations of the related approaches. We specify a list of requirements to be fulfilled by a suitable solution for resource

management in P2P systems. Then, we provide an overview of our contribution to this research topic and conclude the chapter with a road-map of the document.

1.1 Research Statement

The resource management problem in P2P is a combination of two subproblems: the resource distribution and the resource location. The resource distribution subproblem poses the question where to keep the resources of the network when there is no central server. The resource location problem deals with the question where to find the resource when it is requested. These subproblems form the resource management problem that can be simply stated as follow:

After joining the network, a publisher P inserts an item X, say a file, in the system. At a later point in time, a subscriber S wants to retrieve X. The research question is how the network should be organized so that a subscriber S can easily find the location of a server that has a replica of X or that knows the location of P?

One intuitive approach to solve this problem is to maintain a central database that maps a key (e.g. hash of a filename) to the location of servers that store the item. Napster [9] adopts this approach for song titles, but it has inherent reliability and scalability problems that makes it vulnerable when there are attacks on the database. Another approach, at the other end of the spectrum, is for the consumer S to broadcast a message to all its neighbors with a request for X. When a node receives such a request, it checks its local database. If it has X, it responds with the item. Otherwise, it forwards the request to its neighbors, which execute the same protocol. Proceeding in this manner will ensure that

a requested resource is always be found when it exists. However, this solution has some critical limitations like the large overhead produced and the looping problem. Gnutella is a protocol approach for distributed data management that is based on this idea with some mechanisms to avoid request loops. It uses the time-to-live (TTL) flags in the request message to limit the broadcast scope of the message. However, this scoped broadcast approach does not scale either as thoroughly investigated in [84, 98], because of the bandwidth consumed by broadcast messages and the computing cycles consumed by the many nodes that must handle these messages. In fact, the day after Napster was shut down, reports indicated that the Gnutella network collapsed under its own load, created when a large number of users migrated to Gnutella for sharing MP3 music files. To reduce the cost of broadcast messages, several other studies have been proposed in the literature to support intelligently forwarding [123] and directed bread first search [123]. Many variants of the well-known depth first search (DFS) have been also proposed. In the DFS approaches, searches start at the top of the hierarchy and, by following forwarding references from node to node, traverse a single path down to the node that contains the desired data. Directed traversal of a single path consumes fewer resources than a broadcast. Many of the current popular systems, such as KaZaA[5], Grokster [2], and MusicCity Morpheus[6], which are all based on the FastTrack platform, adopt this concept. However, the disadvantage of these approaches, which are considered as hierarchical, is that the nodes higher in the tree take a larger fraction of the load than the leaf nodes, and therefore require more expensive hardware and more careful management. The failure or removal of the tree root or a node sufficiently high in the hierarchy can be catastrophical for the stability of the system.

Structured data management solutions avoid the drawbacks of the above approaches. Searches are performed by following links from node to node until a

provider or a broker node containing the data or the pointer to the data is found. In structured approaches, nodes are self-organized into a virtual construction called overlay so that searches can start at any node and each node is involved only in a limited fraction of the search paths in the system. The problem with structures however, is the cost for maintaining it in a consistent state, when there are churns in the network. In the most outstanding structured P2P solutions, — that we have investigated in more detail in the Chapter 3 on related work, — the size of information kept locally at each node to ensure consistency is high and dependent on the number N of nodes in the system with a complexity of $O(\log N)$. Thus, the length of the maximal path to a requested resource is far from the Moore Bound[1] with a length of $\log N$ in the general cases. Although structured P2P approaches present some interesting properties, there is so far no solution that efficiently resolve the problem of resource management in P2P systems. The recently proposed solutions for structured and unstructured P2P have advantages and severe limitations with regard to performance issues. To design an adequate P2P system, we discuss in the next section a list of requirements that should be fulfilled by a successful resource management solution. We argue that a better P2P performance can be achieved with a more flexible and intuitive architecture which should be well-researched.

[1]See Appendix A for more on graph theoretical background information.

1.2 Requirements for Resource Management Solutions in P2P Systems

As discussed in the previous section, P2P systems are usually large-scale dynamic systems whose nodes are distributed on a wide geographic area. In order to enable resource awareness in such a large-scale dynamic distributed environment, a specific resource management strategy is required that takes into account the main P2P characteristics. Within the scope of this thesis, we argue that a suitable solution for resource management in P2P systems must fulfill the following requirements:

Fault-tolerance: because of the dynamic behavior of its nodes, an appropriate resource management strategy for P2P systems must support fault-tolerance in its operations. In other words, operation such as routing between any two nodes x and y must be completed successfully even when some nodes on the way from x to y fail unpredictably.

Shortest path for lookup operation: the requirement for shortest path of the resource lookup operation is a benchmark for the effectiveness of the resource management. Herewith, any requested resource should be found within an optimal lookup path length that is as close as possible to the Moore Bound $D = \log_{\Delta-1}(N_{max}(\Delta - 2) + 2) - \log_{\Delta-1}\Delta$, [78, 61]. Here, D is the diameter of a Moore graph which is defined as the lowest possible end-to-end distance between any two nodes in a connected graph.

Low cost for network maintenance: the management of a node's insertion or deletion in the network, as well as the dissemination and replication of resources generate control messages in the network. Control messages are mainly

used to keep the topology-changing network up-to-date and in a consistent state. However, since the number of control messages can become very large and grow even larger than the number of data packets, it is of interest to keep the proportion of control messages to the data packets as low as possible. The cost for resource management should not be higher than the cost of the network resource utilization itself.

Load Balancing: the load distribution is measured by investigating how good the network management duties are distributed between the peers in the network. A parameter for assessing this is for example the routing table and the location table at each node of the system. A suitable resource management strategy for P2P should ensure a well-balanced distribution of the management duties between the nodes of the system.

High Availability: the availability of a P2P management solution defines the probability that a resource is successfully located in the system. A resource management strategy is said to be highly available, when it enables any existing resources of the system to be found when it is requested with a probability of almost 100%. This depends on the fault-tolerant routing and the resource distribution strategies.

In summary, a suitable solution for resource management in P2P systems should support fault-tolerant operations, shortest path length to requested resources, low overhead generation during network management operations, well-balanced of load distribution between the peers and high probability of lookup success. In this thesis, we are looking for a resource management solution that can fulfill these requirements the best.

1.3 Research Contributions

In this thesis, we propose HiPeer which is an evolutionary approach to P2P systems that fulfills almost all the requirements of the precedent section. The approach is evolutionary in the sense that it proposes a new step toward the application of P2P into real-time services scenarios. The approach is novel and beneficial from the conception and performance viewpoint. It is a structured P2P concept that enables efficient resource management in P2P systems even during high rate of network churns. Three major contributions to P2P research are provided in this doctoral thesis.

I. A highly connected and easy to manage P2P overlay: the work proposes a novel, real world applicable model for a P2P overlay. Nodes in P2Ps are represented on a so-called concentric multi-ring (CMR) overlay, where each node has an identifier (ID) which is assigned at network joining time by other nodes already in the system. There is no central network manager on the CMR overlay and each node is aware only of a portion of the network, with a routing table limited to a maximal value of $2\Delta + 2$, where Δ is the degree of the De Bruijn network formed by the rings of the CMR architecture. The CMR overlay is highly interconnected because routing between any two nodes on the CMR overlay is always successful as long as there are at least $N = \Delta$ nodes in the network, even when half of the nodes in the network fail.

II. A scalable routing and lookup strategy: based on the CMR topology and on the De Bruijn digraph concept, the work proposes a routing scheme that enables connections between any two nodes in at most $\log_{\Delta}(N(\Delta-1)+\Delta)$ overlay hops. The resource distribution strategy is based on a slightly modified version of the distributed hash table (DHT), where each node publishes its resource to a set

of nodes that have the same prefix (in the ID space) as the published resource. This can be exploited in order to support range queries while preserving load balancing in the network. By coupling the routing and the resource distribution approaches, we propose a resource lookup strategy that enables any resource to be found within at most $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ overlay hops. We prove that the maximal path length to a requested resource D_{CMR} is the lowest bound yet known because it is closer to the Moore bound than the query path length of other outstanding P2P proposals with $2\Delta + 2$ neighbors per node.

III. A low-cost maintenance algorithm: for any node joining or leaving the network, only a local network reorganization is necessary. The number of control packets generated is constant $2\Delta + 2$, allowing a controlled network maintenance cost. In order to detect network inconsistency and to keep the network up-to-date, we deploy a churn-resistant strategy, where each node periodically sends only one KEEPALIVE message to one other node in the system.

Additionally, the HiPeer approach has been thoroughly investigated by means of theoretical analysis and simulation of a prototype Java implementation. Different aspects of the proposed solution have been partly published in [117, 116, 118].

1.4 Structure of the Thesis

The documentation is structured in 8 chapters, each dealing with a particular aspect of the work. As shown in Figure 1.1, the document can be subdivided in 5 parts: (i) introduction, (ii) overview, (iii) approach, (iv) performance evaluation, and (v) conclusion. The current section gives an overview of what the reader will found in each of the next chapters of this thesis.

In Chapter 2: we give answer to two questions:

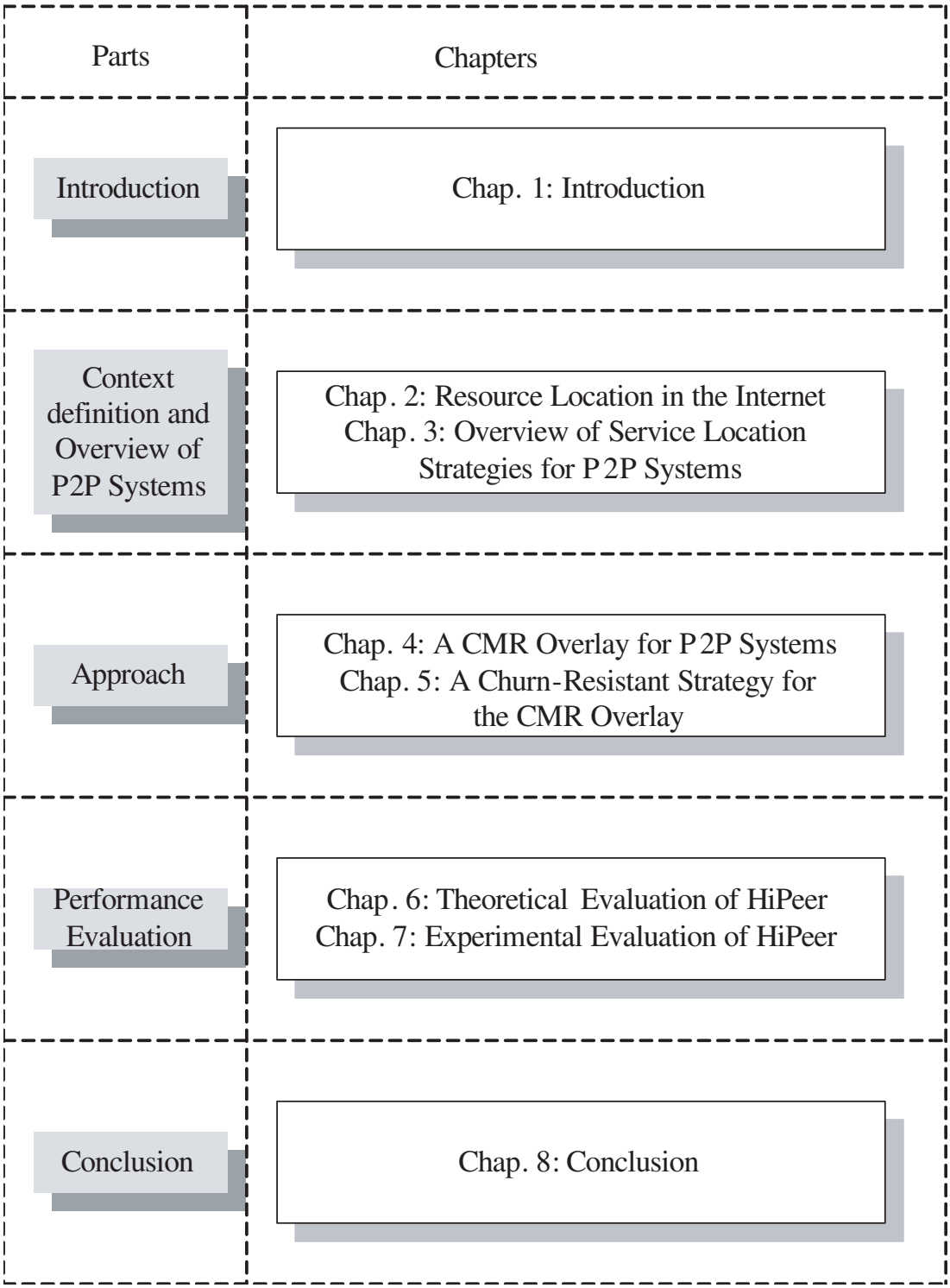

Figure 1.1: Structure of the document

- ◇ why this research work is necessary while considering the plethora of existing resource management strategies in other computer network environments such as the Internet and its edge networks.
- ◇ how the resource management in P2Ps can benefit from the solutions developed in other areas of computer networking.

Here, we discuss some resource management strategies for the Internet core such as the name resolution service (e.g. DNS and DynDNS), the directory service (e.g. LDAP) and other approaches. We investigate resource management solutions in mobile ad-hoc network and in home network environments. We discuss the reasons why the solutions developed there can not be applied to P2P systems and we show that these research solutions constitute an important background for resource management in P2Ps.

In Chapter 3: we present the state-of-the-art on resource management in P2P systems. We begin by classifying the different solutions. We then investigate the solutions of the different groups by highlighting the advantageous and disadvantageous aspects. Finally, we demonstrate through broadly investigation that there is no resource management strategy that meet all the requirements of Section 1.2.

In Chapter 4 and Chapter 5: we describe our resource management approach. Chapter 4 presents the static aspect with the overlay construction, the routing and the lookup schemes. Chapter 5 focuses on the dynamic management aspect. It gives details on how nodes join the system and how the nodes in the system behave when a peer voluntary or ungracefully leaves. We discuss some relevant network churn scenarios such as: (i) what happens when an individual node leaves the network, and (ii) how the system behaves when a collection of nodes (e.g. all nodes on a ring) leave collectively.

In Chapter 6: we analyze the proposed solution. Here, we formally define a dynamic model of the network as a Poisson distribution. We use this model to make the assertion that not more than the half of the network can fail simultaneously in the worst case. Based on this assertion, we show that the network can always self-converge to a consistent state and that a resource can always be found as long as at least Δ nodes are available in the system. Additionally, we investigate some network properties such as load distribution and cost of generated control overhead.

In Chapter 7: we interpret the results of an experimental investigation of a prototype Java implementation of our resource management solution. We use this experimental study to confirm the results of the analysis in Chapter 6. In the simulation, we measure different performance aspects such as the variation of the routing table size, the path length of requests, the number of resources located at each node and the join duration time.

In Chapter 8: we summarize what have been achieved and compare it to the requirements list. This chapter concludes the document by giving some possible extensions and future research topics related to HiPeer.

Chapter 2

Service Location in the Internet

SInce the early days of data communication at the beginning of the sixties, the main motivation for interconnecting two or more computers in a so-called data communication network has been to enable network resources to be exchanged. The network resources represent computer objects such as files, piece of memory space or slice of processing time which can be accessed by other nodes in the network. However, in order to access and to use a network resource, a major concern has been to locate providers of the required resource. One basic example of data communication network is the interconnection of a PC and a printer in a network, where PC applications access the printer service.

As the computer network evolves very fast in the last fifty years to become the Internet today, many different mature solutions for resource management have been in use. These solutions differ from each other according to their level of application in the OSI reference model. They can also be classified by the kind of network architecture they are designed for: centralized or decentralized. In this background chapter, we principally investigate the resource management strategies at the application layer of the OSI reference model. The intention is to export the ideas for resource publishing and discovery in the Internet to the

resource management in P2Ps. The parallels to the agent concept are presented at the end of this chapter.

2.1 Data Communication Network: a Brief History

The Advanced Research Projects Agency Network (ARPANET) developed by the U.S. Department of Defense called ARPA was worldwide the first operational packet switching network in the sixties. ARPANET has evolved to become the global Internet at the beginning of the eighties.

The primary motivation that has led to both networks the ARPANET and the Internet was resource sharing. For example, it was of great interest to allow users on the packet radio networks (PRNET) to access the time sharing systems attached to the ARPANET. The time sharing technique is a computer technique permitting many simultaneous user's accesses to a central computer through remote terminals. The Compatible Time-Sharing System (CTSS) was 1961 the system with the first computerized text formatting utility, and one of the very first to have inter-user electronic mail. Connecting both the PRNET and the ARPANET together was far more economical than duplicating these very expensive computers at that time. However, while file transfer and remote login (e.g. Telnet) were very important applications, electronic mail has probably had the most significant impact of the innovations from that era. Email came with a new model of how people communicate with each other, and it has radically changed the nature of collaboration, first in the building of the Internet itself and later for much of society. There were other applications proposed in the early days of the Internet, including packet based voice communication (the precursor of Internet

telephony), various models of file and disk sharing, and early "worm" programs that showed the concept of agents (and, of course, viruses). A key concept of the Internet is that it was not designed for just one application, but as a general infrastructure on which new applications could be conceived, as illustrated later by the emergence of the World Wide Web (WWW). It is the general purpose nature of the service provided by TCP and IP that makes this possible.

In the 1980s, the widespread deployment of specific network solutions such as the Local Area Networks (LANs), the Personal Computers (PCs) and the workstations becomes important. It was a decisive impetus which helps the nascent Internet to prosper. The data communication world evolves from having a few networks with a modest number of time-shared hosts (the original ARPANET model) to a big network of many networks. This has resulted in a number of new concepts and changes on the management strategies. In order to bring this large network of networks nearer to their users, many researches have been initiated. Hosts were assigned names for example, so that it was not necessary to remember the numeric addresses when requesting resource on a network computer. Originally, there were a fairly limited number of hosts, so it was feasible to maintain a single table of all the hosts and their associated names and addresses. The shift to having a large number of independently managed networks (e.g., LANs and MANET) made it unfeasible to continue to maintain of a single table of hosts, and solution such as the Domain Name System (DNS) was invented by Paul Mockapetris of USC/ISI [86]. The DNS permitted a scalable distributed mechanism for resolving hierarchical host names (e.g. `www.tu-berlin.de`) into an Internet address. In addition, other solutions based on directory servers have been proposed. Today, the Internet has radically transformed the computer and communications world like nothing before. It is at once a world-wide broadcasting

capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers without regard for geographic location. The Internet consists of many isles networks also called edge networks with many specific resource management solutions have been proposed for the many specific parts of the Internet. These solutions are investigated in the following sections.

2.2 Resource Management in the Internet

In its root as the ARPANET, the Internet was conceived primarily as a means of collaboration between many computers for information exchange. Presently, the Internet provides access to thousands of terabytes of online data. However, in order to make efficient use of this wealth of information, users or computer applications need ways to locate information of interest. In the past three decades, a number of resource management strategies have been experimented and improved to efficiently perform over the client-server Internet. We differentiate naming, index, directory services and many other approaches.

2.2.1 Name Resolution Services

The name resolution is the most basic and widely used resource location system in the Internet. Naming is necessary to map easy-to-remember names to physical location of machine (on the Internet, a name-to-IP mapping). Naming is the core operation on any resource management system, and each resource management strategy could in general be considered as a type of name resolution service, since they are trying to map a resource's name with the physical location of its provider. Name resolution services deployed in the Internet to enable resource location has been conceived as distributed client-server systems with dedicated

server and client machines. Generally, many name resolution servers can interact to allow services discovery. Here, we present some examples of name resolution services.

2.2.1.1 Domain Name System (DNS)

The domain name system [86] is a global network of servers that translate host names like `www.tu-berlin.de` into numerical IP (Internet Protocol) addresses, like 130.149.4.134, which computers on the network use to communicate with each other. Without DNS, the user should be memorizing long numbers instead of intuitive URLs or email addresses. And that wouldn't be much fun. Paul Mockapetris designed DNS in 1984 to solve escalating problems with the old name-to-address mapping system. The old system consisted of a single file, known as the host table, maintained by the Stanford Research Institute's Network Information Center (SRI-NIC). As new host names trickled in, administrators of the SRI-NIC added them to the table,— a couple of times a week. Systems administrators would grab the newest version (via FTP) and update their domain name servers. But as the network grew, the host table became unwieldy. Though it worked fine for name-to-address mapping, it wasn't the most practical or effective way to update and distribute the information. And since the stability of the rapidly growing Internet was at stake, Mockapetris along with some other researchers decided to find a better way. The great thing about the domain name system is that no single organization is responsible for updating it. It's what's known as a distributed database; it exists on many different name servers around the world, with no single server storing all the information. Because of this, DNS allows for almost unlimited growth.

DNS software is generally made up of two elements: the name server, and a resolver. The name server responds to application's (e.g. a browser) requests by

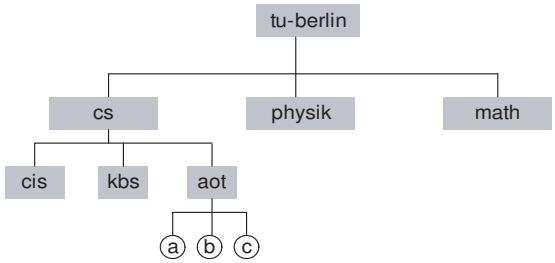

Figure 2.1: A DNS name space example

supplying name-to-address conversions. An application always sends its request to the pre-configured DNS server which is generally in the same domain. If that server has ever fielded a request for the same host name (within a time period set by the administrator to prevent passing old information), it will locate the information in its cache and reply. If the name server is unfamiliar with the domain name, the resolver will attempt to "solve" the problem by asking a server farther up the tree of the domain name space. If that doesn't work, the second server will ask yet another,— until it finds one that can supply the answer. When a server can supply an answer without asking another, it's known as an authoritative server. An exemplary tree representation of the domain name space of the TU-Berlin's network is shown in Figure 2.1. Here, a node *a* willing to access the website of the physik department `www.physik.tu-berlin.de` would subsequently contact the local DNS server of the `aot` domain, the DNS server of the `cs` domain and then the DNS server of the `tu-berlin` if the resource has not been found after the precedent iteration. Once the information is located, it's passed back to the application, and both the requester and the provider can interact. Usually this process occurs quickly, but occasionally it can take an excessively long time (like 15 seconds). In the worst cases, a failure message is returned to the application with the mention that the domain name doesn't exist, even though it is damn well know that it does. This happens because the authoritative server is slow replying to the request, and the communication stack

of the machine gets tired of waiting and it drops the connection. The Domain Name Service is widely used for locating services in the Internet, but even in the static Internet it suffers from congestion problems at the root of the DNS tree. It can not be efficiently applied to P2P resource management because it uses predefined configuration, — of local DNS servers,— at each node.

2.2.1.2 Windows Internet Naming Service (WINS)

The Windows Internet Naming Service (WINS) [12] supports name resolution, the automated conversion of computer names to network addresses in Microsoft Windows networks. Specifically, WINS converts NetBIOS names to IP addresses on a LAN or WAN. Like DNS, the Windows Internet Naming Service employs a distributed client-server system to maintain the mapping of computer names to addresses. Windows clients can be configured to use primary and secondary WINS servers that dynamically update name/address pairings as computers join and leave the network. WINS is nothing other than a Microsoft's implementation of the DNS idea.

2.2.1.3 Dynamic DNS (DynDNS)

The Dynamic DNS (DynDNS) is an approach of DNS for servers with dynamic changing IP addresses. It allows an Internet domain name to be assigned to a varying IP address. This makes it possible for other sites on the Internet to establish connections to the machine without needing to track the IP address themselves. A common use is for running server software on a computer that has a dynamic IP address (e.g., a webserver with a dialup connection, where a new address is assigned at each connection, or a home server with a connection via cable or DSL where the address is changed by the Internet service provider occasionally).

To implement dynamic DNS, it is necessary to set the maximum caching time of the domain to an unusually short period (typically a few minutes). This prevents other sites on the Internet from retaining the old address in their cache, so that they will typically contact the name server of the domain for each new connection. Dynamic DNS service is provided on a large scale by various organizations, which retain the current addresses in a database and provide a means for the user to update it as required. Some "client" programs will, when installed, operate in the background and check the IP address of the computer every few minutes. If it has changed, then it will send an update request to the service. Many routers and other networking components contain a feature such as this in their firmware. Like in P2Ps, DynDNS must manage changes on the node's state such that every node of the network is aware of any other.

Like in the DNS approach, DynDNS requires some static preconfiguration to work properly.

2.2.1.4 Intentional Naming System (INS)

The intentional naming system (INS) [17] is an approach of name resolution that defines the network as a set of components where clients and services are organized in a decentralized network of "Intentional Name Resolvers" or INRs. Clients send requests to INRs, specifying a particular name-specifier, which is matched against the services advertised in the resolver network. Clients periodically advertise the intentional names (INs) of their services to the system to describe what they provide. Intentional names are based on a set of attributes and values (i.e., key/value pairs) that allow expressing generic system information in hierarchical form. The main activity of an INR is to resolve INs to their corresponding network locations. When a request message arrives at an INR, it performs a lookup in its name-tree. The lookup returns information which includes the IP address(es)

of the destination(s) with advertisements that match the requested name as well as a set of routes to next-hop INRs to support routing when mappings change in the middle of a session. INRs also store metric information (such as load, distance, etc) to facilitate self-managing of the different parameters the service is subjected to. Load management, for example, is thus performed by the INRs. While providing certain desirable qualities such as automatic load balancing and self-management, INS is still dependent on network infrastructure that has to be maintained (i.e., the INRs).

2.2.2 Directory Services

Beside the naming resolution services, the concept of directory services has been developed for managing computerized content on directory server computers. The directory itself is the database that holds the information about objects that are to be managed by the directory service. The directory service is the interface to the directory and provides access to the data that is contained in that directory. It acts as a central authority that can securely authenticate resources and manage identities and relationships between them. A directory service works like a phone book. It provides a listing of all parties on a network including email addresses, computers and peripherals like printers and storage devices. A directory service maps the names of network resources to their respective network addresses. Similar to a domain name service, the user doesn't have to remember the physical address of a network resource; providing a name helps locate the resource. Each resource on the network is considered as an object on the directory server. Information about a particular resource is stored as attributes of that object. Information within objects can be made secure so that only users with the available permissions are able to access it.

Directory services were part of an Open Systems Interconnect (OSI) initiative to get everyone in the industry to agree to common network standards to provide multi-vendor interoperability. In the 1980s the ITU and ISO came up with a set of standards - X.500, for directory services, initially to support the requirements of inter-carrier electronic messaging and network name lookup. The Lightweight Directory Access Protocol (LDAP) is based on the services of X.500, but it uses the TCP/IP stack, giving it more relevance on the Internet. There have been numerous forms of directory service implementations from different vendors. Among them are:

- NIS: the Network Information Service (NIS) protocol, originally named Yellow Pages (YP), was the Sun Microsystems implementation of a directory service for Unix network environments. (In the early 2000s, Sun has developed its LDAP-based directory service to become part of Sun ONE, now called Sun Java Enterprise.)

- eDirectory: the Novell's implementation of directory services. eDirectory supports multiple architectures among them Windows, Netware, Linux and several variants of Unix and has long been used for user administration, configuration, and software management. eDirectory has evolved into a central component in a broader range of identity management products. It was previously known as Novell Directory Services.

- Fedora Directory Server: Red Hat released the directory service that it acquired from Netscape Security Solutions under the name: "The Fedora Directory Server".

- Active Directory: Microsoft's directory service is the Active Directory which is included in the Windows XP and Windows Server 2003 operating system

versions.

There are also plenty of open-source tools to create directory services, including OpenLDAP [10] and the Kerberos [74], and Samba software [114] which can act as a Domain Controller with Kerberos and LDAP back-ends.

2.2.3 Search Services

As the Internet is growing very fast, users usually have the problem that they know which information they need but they do not know neither the name of the service nor the name of the provider. In this case, another search step before directory services or name resolution services is required. This is a generic version of range query in P2Ps. For that purpose, more generic and open ended search services on the Internet have become popular with the advent of search engine at the beginning of the 90s such as *google*, *altavista* and *yahoo*. A search engine is a program designed to help finding information stored on a computer system such as the World Wide Web, or a personal computer. The search engine allows for asking for specific contents that meet specific requirements and retrieves a list of references that match the specified requirements. Search engines use regularly updated indexes to operate quickly and efficiently. Without further qualification, search engines usually refer to as web search engines, which search for information on the public web. Other kind of search engines are enterprise search engines, which search on the Intranet and personal search engines, which search individual personal computers. For example, printing could be considered a special case of Intranet search, since a print operation on a nearby printer can be started using the appropriate search parameters. In large organizations, where employees often move between different offices or buildings, the use of search allows them to locate particular items of information, or the resources needed for particular tasks.

Search services differ from name resolution and directory services in that it is more open-ended. At the end of a search query, the user will commonly perform a final operation either of name resolution, or directory service request to obtain the location or information about the resource requested, respectively. Therefore, depending on the way it is implemented, search can be considered either as a component at the same level of directory services and name resolution, or as a middleware built on top of them. The search services generally function based on an index server that maintains keywords and key information and so enabling to do a key/server mapping, when necessary.

2.3 Internet Edge Networks

Along with the resource organization strategies described in the last section designed for the Internet core, a range of network specific solutions have been proposed recently to address the challenging issue of resource location at the edge of the Internet. We consider resource management solutions in MANETs (Mobile Ad Hoc Networks), home and enterprise networks. These networks at the edge are different from the core Internet with their size and the communication link (wired or wireless), etc. MANETs are self-managed without any central administration instance, and with dynamic topology like P2P. Home networks are generally small networks with high real-time requirements for automatic reconfiguration and automatic service location.

2.3.1 Resource Management in MANETs

Resource management in mobile ad-hoc network (MANET) has been up for more than thirty years now. A mobile ad-hoc network is a collection of mobile nodes

interconnection through wired or wireless connections without any central management instance. MANETs are mainly stand-alone network that can be connected to the other networks like the Internet by using gateways. The first generation of MANET goes back to 1972. At the time, they were called PRNET (Packet Radio Networks). In conjunction with ALOHA (Areal Locations of Hazardous Atmospheres) and CSMA (Carrier Sense Medium Access), approaches for medium access control and a kind of distance-vector routing in PRNET were used on a trial basis to provide different networking capabilities in a combat environment. The second generation of ad-hoc networks emerged in 1980s, when the ad-hoc network systems were further enhanced and implemented as a part of the SURAN (Survivable Adaptive Radio Networks) program. This provided a packet-switched network to the mobile battlefield in an environment without infrastructure. In the 1990s, the concept of commercial ad-hoc networks arrived with notebook computers and other viable communications equipment. The resource management issue in MANET has been focused on how to route between any two mobile nodes to enable information exchange.

A range of different routing strategies have been proposed recently to enable resource management at the network layer of the network [102]. The different approaches are generally classified using different criteria. They can be classified based on the manner forwarding decisions are taken. We differentiate: (i) distance vector routing using distance or delay to a node to take routing decision [92, 91] (ii) link state routing using information about state of the outgoing links [51, 37], (iii) position-based routing using geographical positioning devices or other techniques [73, 103, 66, 71], and (iv) hybrid routing schemes combining two or more of the precedent strategies [31]. Another classification criteria is the virtual organization of the topology. Some MANET proposal such as [120, 33] are based on hierarchical topology, where nodes in the network are whether normal or super

nodes. Other algorithms are based on flat topology.

Routing in MANET defines an interesting background for the work in this thesis. MANET can be considered as a special case of P2P systems. The resource management problem in P2P is more generic than in MANET and can be deployed on top of any network such as the Internet spanning a network layer over the physical network and the routing layer.

2.3.2 Resource Management in Home Networks

Along the years, home networking has evolved to a wired and wireless data communication network between PCs, audio/video devices, IP telephone and household devices such as washing machine and fridge. The objective is to enable ambient intelligent environment at home, where devices can interact with each other and provide seamless service access facilities for user in the household. The user should not be aware of the devices and the services should be set in the foreground. One challenging aspect to be addressed in order to meet these objectives is the service management aspect, which has been thoroughly studied in the related literatures. We have identified a number of novel and interesting approaches that has focused on strategies for service advertising, discovery and access in a dynamic distributed home network environment. Roughly, they are based on the main idea that a device coming online or joining the home network advertises its services or listens for advertisements of available services. It sends a resource request to a provider or a broker when willing to use the service of another network participant. Here, we present some architectures and schemes related to service advertisement, discovery and access in home networks.

2.3.2.1 Service Location Protocol (SLP)

The Service Location Protocol (SLP) [56] allows computers and other devices to find services in a local area network without prior configuration. SLP has been designed to scale from small, unmanaged networks to large enterprise networks. SLP is based on the directory service concept to store and retrieve network services; but SLP can also work without a directory server called a directory agent. A node in an SLP network can play three different roles:

- ◇ User agent (UA): when it is looking for services in the network.
- ◇ Service Agent (SA): when it joins the network and publishes one or more services.
- ◇ Directory Agent (DA): when it plays the role of a device that caches service information. DAs are used in larger networks to reduce the amount of traffic and to allow SLP to scale. The existence of DAs in a network is optional, but if a DA is present, UAs and SAs are required to use it instead of communicating directly.

SLP is designed to support P2P communication between the nodes. The operation of SLP differs considerably, depending on whether a DA exists in the network or not. When a client first joins a network it multicasts a query for DAs on the network. If no DA answers it is assumed that there is no DA in the network. It is also possible to add DAs later, as they multicast a "heartbeat" packet in a predefined interval that will be received by all other devices. When a SA discovers a DA, it is required to register all its services at the DA. When a service disappears the SA should notify the DA and unregister it. In order to send a query in a network without DA, the UA sends a multicast packet that contains the query. All SAs that contain matches will send an answer to the UA. If the answer is too

large to fit into a single UDP packet, the packet will be marked as "overflown" and the UA is free to send the query directly to the SA. In order to send a query in a network with DA on the other hand, the UA will send the query packet to the DA. As every SA must register all services with the DA, the DA is able to fulfill the request completely and simply sends the result back to the UA. The idea behind SLP with DA has been adopted in pseudo P2P example like Napster. One disavantage of the SLP without DA is the flooding property of the multicast operation, which is undesirable in large systems.

2.3.2.2 SSDP/Universal Plug-and-Play

The simple service discovery protocol (SSDP) [52] is the service location protocol of the Microsoft's Universal Plug and Play (UPnP) initiative. It is oriented toward home networks. Like in SLP, SSDP enables devices to request information about services on a network and to advertise their presence and their offered services.

A UPnP network consists of UPnP devices that publish their services to a UPnP control point. The UPnP control point is a central registry point such as the directory agent in SLP. To enable seamless access to services, the UPnP protocol consists of five basic operations: description, discovery, control, event notification and presentation.

When a device is added to the network, the UPnP discovery operation allows that device to advertise its services to control points on the network. Similarly, when a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. The fundamental data exchange in both cases is a discovery message containing a few, essential specific infos about the device or one of its services, e.g., its type, identifier, and a pointer to more detailed information (an URL). After a control point has discovered a device, the control point still knows very little about the device. For

the control point to learn more about the device and its capabilities, or to interact with the device, the control point must retrieve the device description from the URL provided by the device in the discovery message. After a control point has retrieved a description of the device, the control point can send actions to a device's service. To do this, a control point sends a suitable control message to the control URL for the service (provided in the device description).

A UPnP description for a service includes a list of actions the service responds to and a list of variables that model the state of the service at run time. A device publishes updates when these variables change, and a control point may subscribe to receive this information. The service publishes updates by sending event messages containing the names of one of more state variables and the current value of those variables. UPnP uses a simple mechanism for accessing device's services in a home networking environment in a seamless manner. Other than in SLP, the UPnP protocol does not foresee service discovery without the intervention of one or more control points. UPnP uses a centralized approach for service publishing and discovery which is inappropriate for P2P; although it shows highly availability and stability in small client-server networks.

2.3.2.3 Bluetooth and the Service Discovery Protocol (SDP)

SDP [7] is the service discovery protocol defined in the Bluetooth specification to address the unique characteristics of the Bluetooth environment. In Bluetooth network, a set of available services changes dynamically based on the radio range of the devices in motion. The whole purpose of the SDP is to allow Bluetooth devices to discover what other Bluetooth devices can offer (services for example). SDP allows this in various means. "Searching" is used to look for specific services, while "browsing" is used for searching services actually being offered. Bluetooth is based on the client-server principles where there is a node playing the role

of master and others playing the role of slaves. After joining the network, the client registers its services with the master node. On client request, the master responds with positive notification when the requested service is available and the client can directly connect to the service provider in order to access the desired information.

SDP is an interesting service discovery approach designed for small personal area networks (PAN)s with Bluetooth nodes. It is based on the client-server concept and uses P2P only for service access between clients exchanging data.

2.3.2.4 Salutation

The Salutation architecture is a service discovery and service management product from the Salutation Consortium[1], a nonprofit corporation with contributions from IBM. Salutation is an open standard, independent of operating system, communications protocol, hardware platform, or vendor-imposed limitations. It was created to provide service discovery for a broad range of network appliances and equipment in a platform-, OS-, and network-independent environment. Devices can use it to advertise and describe their capabilities and discover the capabilities of other devices by using publish and search features.

The Salutation architecture is composed of two major components: Salutation Manager and Transport Manager. The Salutation Manager (SM) is the core of the architecture, similar to the directory agent in Service Location Protocol. It is defined as a service broker; a service provider (a device) registers its services in a SM. When a client asks its local SM in the LAN for example for a service search, the search is performed by coordination among SMs. Then, the client can use the returned service. A SM sits on top of the Transport Managers that provide reliable communication channels, regardless of what the underlying network

[1] `www.salutation.org`

transports are.

Salutation proposes a scalable approach for service management in a client-server environment. It is limited because of its centralized server (SM).

2.3.2.5 Jini

Jini [13] is a Java-based technology defined by Sun Microsystems. When Jini-enabled devices connect to a network, they establish impromptu Java-oriented network that let users immediately access network resources and services. The technology is designed to support any device that "passes digital information in or out" according to Sun. Devices register with a registry service in the network when they connect, which makes them available to other devices. For example, when a printer is attached and gets registered, it makes its driver available on the network and this driver gets downloaded to clients when they need to use the printer. Based on Java technology, Jini technology uses Java Remote Method Invocation protocols to move code around the network.

However, the approach is limited because it uses centralized service registry instance to enable resource discovery. Moreover, it relies on Java-RMI which also limits its application scope.

2.4 Agent Technology for Service Location in Distributed Systems

Peers in a P2P system have many similarities with agents in agent technology. The word "agent" in computer sciences and within the scope of artificial intelligence has been defined in [18] as a piece of software acting on behalf of a user. The agent performs different tasks according to a well-defined goal to achieve.

In the agent technology viewpoint, an agent can use resources from other network entities and can cooperate with other agents within an agent platform to achieve an individual or a common goal. Like peers in P2P systems, agents are autonomous in their actions and decisions. They can act in groups with or without any central management entities. Hence, like in P2P connected networks, one of the main problem of the agency concept is the resource location.

Some international research and industrial institutions like OMA[2], FIPA[3] or Agentcities[4] have spent a lot of efforts to provide standards that define interfaces between agents and agent platforms. They have proposed solutions for resource location based on the centralized management concept with one or more agents playing the role of management server (AMS) and some others playing the role of directory facilitator (DF). In this concept, AMS and DF are responsible for coordination between agents and used for facilitating access to remote provided services [112]. FIPA started a Technical Committee (TC ad hoc1) in 2002 to develop a solution for FIPA compliant agent platforms in ad hoc environments. Several proposals were made (e.g. [90], [95] and [19]) to enhance the centralized client-server management concept toward a P2P one. The proposed P2P approaches were designed to support self-coordination without DFs. An exemplary application of the P2P version of the FIPA specification for the telecommunication area have been implemented and presented in JIAC IV project [19]. The platform implemented within the JIAC IV project is used for managing a handful of agents in telecommunication applications and it highlights the value of agency based on the P2P concept.

The P2P resource management solution we are researching in this thesis should be able to manage computer systems with larger number of nodes than the

[2] `http://www.oma.org`
[3] `http://www.fipa.org`
[4] `http://www.agentcities.org`

number of agents on a multi-agent platform. With the results of this dissertation, one could imagine interconnecting many agent platforms by means of a scalable P2P management solution, with automatic distribution and discovery of services.

2.5 Summary

We have investigated strategies that are used in the core and at the edges of the Internet to locate resources or services. Over the past years, the Internet has made considerable progress to become a mature technology. Many resource management strategies we have presented here are available as commercial or experimental products. However, although they are undoubtedly interesting management concepts, they are mainly based on client-server architectures. For instance, resource management solutions for home networks were designed to address the problem of service location in a small network environment with a limited number of devices.

Nevertheless, the studied approaches present important foundations for the resource management in self-managed dynamic distributed systems such as P2Ps. Besides, resource management approaches in MANETs present interesting concepts that can be flattened from specific features in order to be used as generic concepts for P2P systems.

Chapter 3

An Overview of Resource Management Strategies for P2P Systems

With the years, the Internet has evolved toward one of the most important technologies which has revolutionized the people's way of life like no other technology in the past. In the same impetus, the resource management problematic has been aroused great attentions, with many solutions as we have shown in the previous Chapter. However, the solutions for resource management in the Internet have been mostly based on the client-server communication concept with static distributed servers for the network management.

In fact, the P2P paradigm comes with a new communication concept that should help where client-server systems have failed. In contrary to the client-server model, each node in a P2P network is designed to assume as much responsibilities as any other. Additionally, each node can simultaneously play the role of client, server or gateway in the system. The P2P concept does not require the

presence of any central administration server in order to manage the communication and the interaction between the nodes. Nodes organize themselves in a network and each must ensure with its local action that the network is always in a consistent state. This new way of communicating presents several advantages such as the fast and easy deployment of communication systems, their cost effective management and the communication without censorship and boundaries.

Applications of P2P range from resource sharing and aggregation in large-scale grid environment [21, 11], groupware and collaboration tools, instant messaging and Internet telephony, to Internet-scale operating systems as envisioned by Anderson & Kubiatowicz in [20].

However, in order to resolve the problem of resource management in P2P systems, many solutions have been suggested recently by research institutions and industries. Nevertheless, this problem remains a highly challenging research issue. The proposed solutions can be classified in two big groups depending on the manner the nodes choose their neighbors and how data are distributed among the nodes within the network. One can differ between: (i) structured P2P architectures, with predefined rules for neighbors and data assignment, and (ii) unstructured P2P architectures that use no configuration rules to specify the interaction between nodes in the system. In this chapter, we review the different classes of architectures and discuss their strengths and limitations. In our investigation, we focus on the following aspects of P2P systems:

- ◇ the routing or search strategy defines and specifies the criteria used for the multi-hop forwarding operation at each node.

- ◇ the logical network constellation or the overlay construction mechanism defines and specifies the topology of the logical network representation. In fact, the P2P communication concept is a generic one; it spans an overlay

on top of the routing or the physical layer of a network such as the Internet to enable location of application-layer resources. Overlays can be designed as geometric figures such as ring, hypercube, toroid, etc.

- ◇ the churn management strategy defines and specifies the control and recovery mechanisms to resolve the problem of network dynamics. In fact, each node joining or leaving the system requires a set of management operations to make its presence or absence aware to the rest. Depending on the type of logical constellation of the overlay, churns provoke high or less network fluctuation and perpetual topology changes.

For each of the studied classes of P2Ps, we investigate their performance and check how good they meet the requirements of Section 1.2.

3.1 Unstructured P2P Architectures

In an unstructured P2P architecture, no rule exists that defines where data is stored and which nodes are neighbors. To find a specific data item, early works used pure flooding, which functions like the Breadth First Search (BFS) on a graph with depth limit D. The parameter D is the system-wide maximum time-to-live (TTL) of a message in terms of overlay hops. In flooding-based approaches, the querying node sends a request to all its neighbors. Each neighbor processes the query and returns the result if the data is found. If a neighbor receiving the request can not provide the requested resource, it forwards the query message further to all its neighbors except the querying node. This procedure continues until the depth limit D is reached. Flooding tries to find the maximum number of results within the ring that is centered at the querying node and has the radius: D-overlay hops. However, it generates a large number of messages (many of

them are duplicate messages) and does not scale well. Gnutella [8] was based on scoped-flooding to limit the problem of looping in original flooding. With scoped-flooding, each message is flooded only to the nodes within a given fixed distance from the source. The distance is given by a TTL flag that defines the broadcasting scope.

Many alternative schemes have been proposed to address the problems of the original flooding roughly based on selective forwarding. These works include iterative deepening [123], k-walker random walk [81], modified random BFS [69], directed BFS [123], intelligent search [69], local indices based search [123], routing indices based search [39], attenuated bloom filter based search [97], adaptive probabilistic search [115] and dominated set search [35]. In the iterative deepening and local indices, a query is forwarded to all neighbors of a forwarding node. In all other schemes, a query is forwarded to a subset of neighbors of a forwarding node.

The searching schemes in unstructured P2P systems can also be classified as deterministic or probabilistic according to how the next hop of a lookup message is chosen. In a deterministic approach, the query forwarding is deterministic, relying on metrics. In a probabilistic approach, the query forwarding is probabilistic, or relying on ranking. The iterative deepening, local indices based search [123], and the attenuated bloom filter based search [97] are deterministic. The other approaches such as the modified random BFS [69] and the adaptive probabilistic search [115] are probabilistic.

Another way to categorize searching techniques in unstructured P2P systems is the differentiation between regular-grained and coarse-grained search schemes. In a regular-grained approach, all nodes participate in query forwarding. In a coarse-grained scheme, the query forwarding is performed by only a subset of

nodes in the entire network. Unstructured P2P architectures based on dominating sets are coarse-grained because the query forwarding is performed only by the dominating nodes in the connected dominating set (CDS). All other approaches are regular-grained. Another taxonomy is blind search or informed search [115]. In a blind search scheme, nodes do not keep information about data location. In an informed search scheme, nodes store some metadata that facilitates the forwarding decision. Blind search approaches include iterative deepening, k-walker random walk, modified random BFS, and two-level k-walker random walk. All other approaches belong to the informed search category.

In the following, we discuss some resource management strategies in unstructured P2P architectures which are relevant to this work.

3.1.1 Dedicated Search Server or Napster Approach

The Napster network [9] is the first really large-scale peer-to-peer infrastructure deployed over the World Wide Web basically for music (MP3) file-sharing. In fact, the Napster network is based on a centralized server, with which each network participant is directly connected as shown in Figure 3.1. After joining the Napster network, a node publishes its resources on the central directory server. The centralized server is aware of all available network resources after the update process.

When looking for a resource in the network, a node sends a query to the central directory server which returns the address of one or many providers of the requested resource. The requesting node interacts directly with the resource provider to access the requested resource as shown in Figure 3.1. Napster enables very fast resource location in term of lookup path length; in fact only one hop

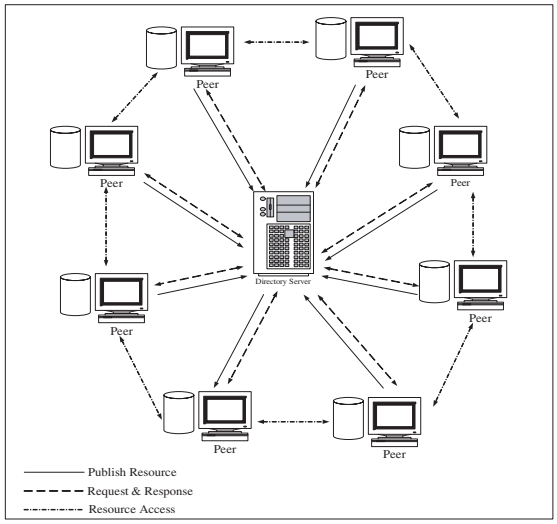

Figure 3.1: The message flow in a Napster network. A client node publishes resources to the centralized directory server and interacts directly with the resource provider to access a resource.

is needed between the requester and the server for locating a provider of a desired resource. However, when the number of nodes N and hence the number of resources on a Napster network grows, the time needed by the server to select a given provider for a requested resource undesirably increases as a linear function of N. Besides, the central directory service is a single point of failure and, the deployment of Napster requires servers that are expensive to build and to maintain. Additionally, Napster is not considered as a pure P2P approach, since it is primary based on the client-server principle during the discovery operation.

3.1.2 Blind Flooding Search or Original Gnutella

The blind flooding mechanism relays the query message to all its logical neighbors, except the incoming peer. This mechanism is also referred to as breadth-first search (BFS) and is used among peers in the original Gnutella or among supernodes in the FastTrack protocol. Gnutella [8] is a distributed software project

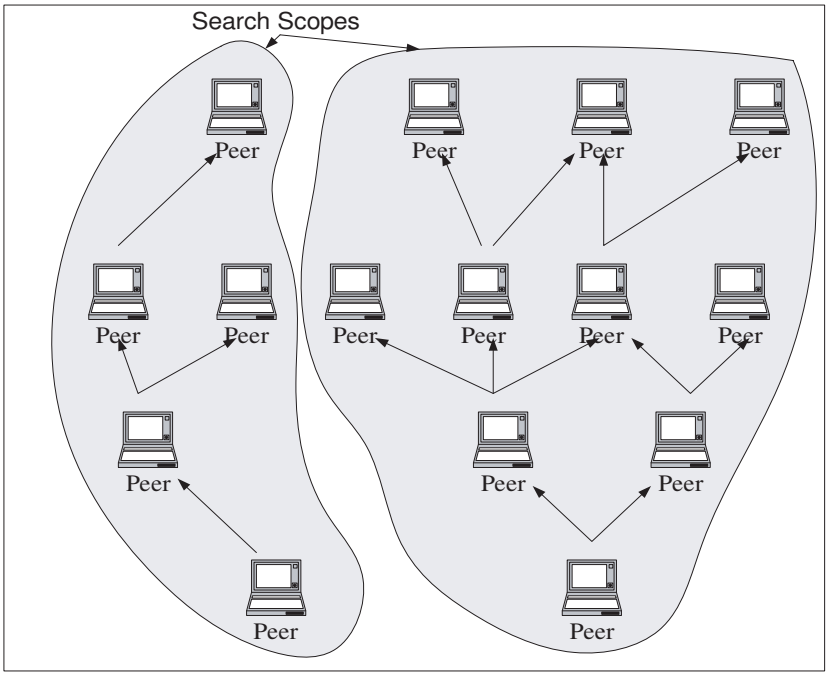

Figure 3.2: Search in a Gnutella network. Client node uses TTL to limit the dissemination scope of the request

aiming at the creation of a true peer-to-peer file sharing network without a centralized server as in former approaches such as Napster. Nodes in Gnutella broadcast requests to their neighbors and use the TTL flag to limit the scope of the request and hence spare bandwidth. When the TTL value is optimally chosen, an existing resource can be found with high probability (w.h.p.). Figure 3.2 shows an exemplary Gnutella network where the scope of request propagation is highlighted. However, flooding-based request propagation is indeed robust but unpracticable in large-scale P2P systems. With blind flooding, each request generates a very large load for each peer with a complexity of the magnitude order $O(\Delta^{TTL})$, where the parameter Δ is the number of routing entries per node. This load exponentially increases with the time-to-live value TTL and with the rarity of the requested resource.

3.1.3 Directed BFS and Similar Approaches

In the directed BFS approach [123] the querying node sends the query message to an intelligently selected subset of its neighbors, — the so-called "good" neighbors,— that are expected to quickly return many high-quality results. These neighbors then forward the query message to all their neighbors just as in original BFS. To choose "good" neighbors, a node keeps track of simple statistics about its neighbors. It uses information such as the communication latency and the number of query results returned through that neighbor. Based on these statistics, the best neighbors can be intelligently selected using the following heuristics: (i) the highest number of query results returned previously, (ii) the least hop-count (i.e. the closest neighbors) in the previously returned messages, (iii) the highest message count (i.e. the most stable neighbors), and (iv) the shortest message queue (i.e. the least busy neighbors).

By directing the query message to just a subset of neighbors, directed BFS can reduce the routing cost in terms of the number of routing messages. By choosing good neighbors appropriately, this technique can improve the quality of query results and lower the query response time. However, in this scheme only the querying node intelligently selects neighbors to forward a query. All other nodes involved in a query processing still flood the query to all their neighbors as in BFS. Therefore, the message duplication is not greatly reduced. Moreover, if the algorithm fails to choose the right "good" node, the resource will never be found.

The "routing indices based search" presented in [39] is similar to the directed BFS in the sense that all of them use the information about neighbors to guide the search and choose the next hop. Directed BFS only applies this information for selecting neighbors of the querying source (i.e. the first hop from the querying

source). The rest of the search process is just as that of BFS. The "routing indices based search" however, guides the entire search process from the source to the destination. This approach stores information about the objects, documents and the number of documents stored in neighbors. The routing indices approach considers content queries, i.e. queries based on the file content instead of file name, file identifier or node identifier. An example of such a content query is the request for documents that contain the word "P2P". The query includes a set of subject topics. Documents are classified in different categories by means of keywords and a document might belong to more than one topic category. Each node maintains a local index of its own document database based on the keywords contained in these documents. Although the "routing indices based search" can enable coarse-grained resource location based on the content, it can not guarantee the resource location.

3.1.4 Hierarchical BFS

Hierarchical approaches of BFS such as FastTrack have been proposed as an optimized alternative to existing flat BFS approaches. FastTrack extends Gnutella to the notion of nodes' differentiation to improve scalability. It defines special nodes called supernodes, which are trustworthy nodes with fast and stable network connection and large computing power. Supernodes are used as connection gateways and indexing servers by other client nodes, when requesting resources in the network. A new client in the network attempts to contact any supernode, and as soon as it finds a working one, it requests a list of currently active supernodes to be used as gateway for future resource requests. The client randomly picks one supernode to which it uploads its list of shared files or objects. Figure 3.3

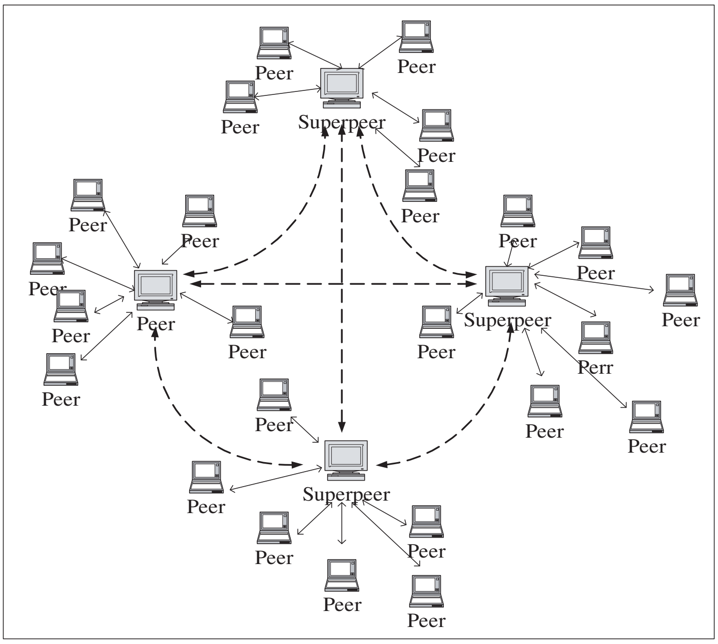

Figure 3.3: The network constellation of a FastTrack file-sharing application. Client nodes connect to a supernode to distribute and to request desired resources.

presents the constellation of a FastTrack network, where each client node is connected with a supernode. For requesting an object, a node sends a request to one supernode to look up for resources in the network and the supernode communicates with other supernodes in order to satisfy search requests. The lookup operation and the overhead generated is less than in Gnutella since the messages are forwarded only among the supernodes.

In the class of unstructured P2P systems, FastTrack is widely represented with several applications such as Kazaa [5], Grokster [2] and iMesh [3], which presents some performance optimization with regard to the robustness in file-sharing in comparison to applications based on Gnutella like the gtk-Gnutella client, Limewire and EDonkey2000. Another open service framework like Jxta [4] is also related on the FastTrack protocol for resource management. However, the FastTrack protocol uses a kind of structured flooding, whereas it can not be guaranteed that a requested resource will be found even when available in

the network. Additionally, the node differentiation concept requires a fair user community because some users will step down their upload speed to avoid the duty of supernodes. Thus, the role supernode is a synonym for more bandwidth consumption and more delay by own data exchange. Worthy to mention is that when the number of supernodes tends to 1, the FastTrack protocol is similar to Napster.

Some other interesting approaches of unstructured P2P architectures relying on hierarchical BFS use the concept of connected dominating set [35, 127, 40]. In the connected dominating set approaches, the nodes in the system are self-organized into two responsibility levels. The nodes of the higher level form what is called the connected dominating set (CDS). A CDS in a P2P network is a subset of nodes which are interconnected through direct overlay hops or virtual links. All other nodes that are not in the CDS can be reached from some nodes in the CDS in one overlay hop. The search scheme in these approaches relies on routing indices that are stored in the nodes belonging to the CDS. Resource location here is achieved by routing a resource request message between the nodes of the CDS until a node of the CDS is found that knows a provider of the requested resource. P2P systems based on CDS have the advantage that flooding of the whole network is avoided. However, the nodes of the CDS are heavily burdened with the management duties than others. Besides, these approaches are characterized by the high cost of creating and maintaining the CDS, when nodes join and leave the network continually.

3.2 Strictly Structured P2P Architectures

Facing the limitations of unstructured P2P architectures, researchers have proposed as alternative strictly structured approaches of P2P architectures, in which

the neighborhood relationship between the peers and the data distribution and replication are strictly defined. The objective is to enable more deterministic resource location operations with shortest lookup path, fault-tolerant operation and high availability. The resource location in structured P2P architectures is determined by a specific application-layer routing scheme according to a virtual network called overlay[1]. The overlay is built on top of the underlying IP routing architecture and specifies the neighborhood relationship between the nodes.

Among the strategies for resource distribution in strictly structured P2Ps, the distributed hash table (DHT) is one of the most popular ones. We differ between strictly structured P2P architectures that are based on DHT and non DHT-based P2P approaches. The DHT concept maps hashed data items into a series of distributed hash tables. In fact, each piece of data item in DHT-based P2Ps is assigned to a unique node, which is responsible for it in the network. Each node is responsible for a certain number of keys, where a key is the hashing result of a data item denoted as: $key = hash(data)$. This means that the responsible node stores the key and the data item or a pointer to the data item associated with that key. To lookup for a resource in a DHT-based approach, a querying node uses the same hash function as the one used by the provider to store the data item in the network. The querying node must also know the parameter of the data item (e.g. the name of a file) to be able to retrieve a possible server responsible for the data item. Data organization schemes based on DHT are typically designed to scale to large number of nodes. Many proposals of P2P systems such as Chord [113], Pastry [99], Tapestry [126], Content Addressable Network [94] and many others [85, 82, 68, 14, 128, 83] are based on flat DHT. However, the capabilities of DHT P2Ps to enabling load-balancing and scalability are overshadowed by its weak locality-awareness and its limitation to supporting only exact matching

[1]Strictly said, CDS in unstructured P2P can be considered as overlays.

of queries; range matching of queries is not supported in the primary version of DHTs. To tackle the issues of locality-awareness noticed in flat DHT-based P2Ps, hierarchical approaches of P2P systems have been proposed. Some of these approaches are HIERAS [121], Cyclone [23], BYPASS [77], Jelly [63] and Grapes [110]. While they are good for ensuring locality-awareness, hierarchical P2P concepts have the shortcoming that they can not guarantee load balancing. In fact, the nodes at the top of the hierarchy are more burdened with the network management task than other nodes at the lower levels.

The non-DHT based P2P approaches such as SkipNet [59], SkipGraphs [24] and their variants [25, 60, 124, 67, 38] aim at solving the shortcomings of DHT-based P2Ps by avoiding hashing. Hashing does not efficiently keep data locality and is not amenable to range queries. When hashing is combined with hierarchy, the load balancing is difficult to achieve. Moreover, systems such as Chord, Pastry and Tapestry have two disadvantages: they provide no control over where data is stored and no guarantee that the routing path remains within an administrative domain.

Moreover, strictly structured P2P systems are based on different network overlay topologies in order to support fault-tolerant and shortest-path lookup operations. The overlay topology can generally be represented as ring [113, 99], tree [93, 99, 126], hypercube [94, 107, 76], and any other graph concept like De Bruijn [47, 87, 48, 27, 80, 68] and shufflenet [82, 76]. Some other topologies called "hybrid" combine two or more of the basic topologies for optimized routing in the network [69, 26, 49, 104, 46, 55, 15, 58].

In the following sections, we study some approaches of strictly structured P2P architectures in more details. Here, we give some functional details on Chord [113], Pastry [99], Tapestry [126] and Content Addressable Network (CAN) [94]. The choices are motivated by the fact that they are closely related to our work.

Readers interested in a detailed review of P2P systems might refer to [22].

3.2.1 Chord

Chord [113] uses a circular overlay to represent the nodes of the network. It is founded on the concept of consistent hashing [70] to assign responsibility of data items to nodes in the network. As claimed in [70] consistent hashing can allow allocation of keys to nodes evenly in the network. In fact, according to the analysis presented in [70], in a network of N nodes and an overall number of K keys, the consistent hashing technique can distribute the keys such that a proportion of $\frac{N}{K}$ keys pro node is approximated.

In Chord each node has a unique decimal identifier (ID) and keeps a finger table that contains their direct (one overlay hop) neighbors on the identifier circle. The finger table of a node with ID x is formed of the nodes $y = x + 2^i$ with $0 \leq i < \log_2 N$, N being the maximum possible number of nodes that form the network. For a given value of i, if the node $y = x+2^i$ does not exist in the network, the entry of the finger table of x corresponding to y is replaced by a node with identifier next larger than y on the identifier ring. Let's explain this by using the example of the Chord ring depicted in the Figure 3.4(a) with $N = 64$ nodes; when the network is fully constructed, a node with identifier 8 has a finger table pointing on the nodes $9, 10, 12, 16, 24$. In the network situation in Figure 3.4(a) the node 8 has a finger table with pointers to the nodes $14, 21, 32, 42$. Additionally to the finger table, each node maintains a link to its predecessor (counterclockwise) on the ring. A data item with a key k is replicated to a node whose identifier is equal to k or next larger clockwise from k on the identifier circle.

When a querying node A is looking for a key k in the Chord network, it forwards the query for k to the node denoted as $successor(k)$. Here, $successor(k)$

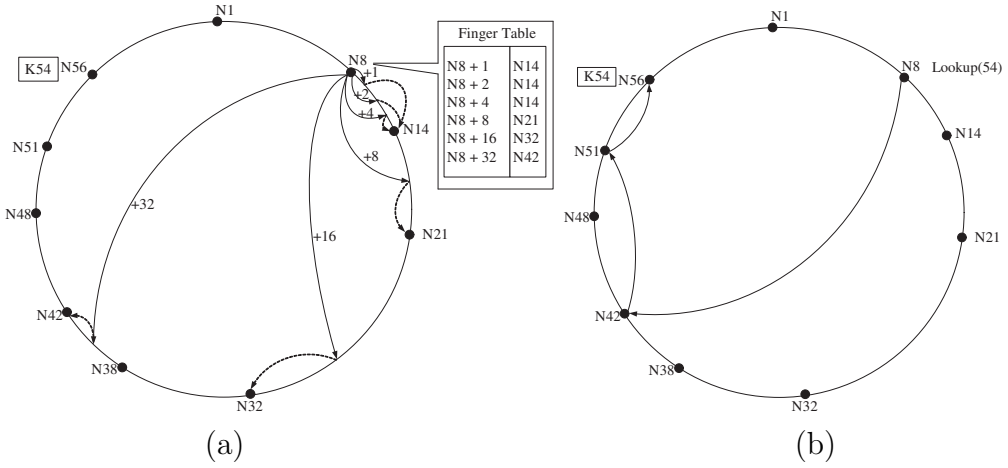

Figure 3.4: Management operation on the circular Chord overlay: (a) the finger table of the node 8, and (b) the lookup operation for a key 54 from the node 8.

is the node of the finger table with identifier equal to k. If the node with identifier k is present in A's finger table, it is reached directly with one overlay hop from A. Otherwise, the query is forwarded to the node in A's finger table that has the largest ID smaller than k in the identifier space. In this way, the query for a key k is forwarded through the successors' list until the node responsible for k is reached. The finger table is used to speed up the lookup operation. In this way, any lookup operation in Chord can be achieved within $\log N$ overlay hops.

To ensure an always consistent state of the Chord ring, each node periodically runs a stabilization routine to maintain both the successor and the finger tables in a consistent state. According to the fact that the entries of the finger table are pointers on nodes that are widely distributed across the network, the stabilization routine will spread the control messages in all parts of the network. This could negatively affect the entire network, causing a high amount of control overhead that can deteriorate the network's performance. A query that is sent while a node is joining or leaving and before the stabilization process is completed, could fail or the lookup path could be larger than $\log_2 N$ overlay hops. We can conclude this investigation of Chord with the observation that its maintenance strategy

is expensive and it achieves $\log_2 N$ overlay hops for routing with a table size of $O(\log N)$ per node.

Some works such as [122] and [50] has been proposed that extend the original Chord approach by adding different kinds of reverse edges. They aim at enabling a modified Chord that is resilient to routing attacks with a lookup path length that is around the $\log N$ range even in failure situations.

3.2.2 Pastry and Related Approaches

The Pastry system [99] is a self-organizing P2P overlay network of nodes which organizes its nodes on a flat ring like in the Chord protocol, [113]. It uses DHT for attribution of identifier to node and to assign a resource's responsibility to a specific node. Each node in the Pastry overlay network has a 128-bit identifier (ID). The ID is used to indicate a node's position in the circular ID space, which ranges from 0 to $2^{128}-1$. The ID is assigned randomly when a new node joins the system. It is assumed that IDs are generated such that the resulting IDs are uniformly distributed in the 128-bit ID-space. For instance, IDs could be generated by computing a cryptographic hash of the node's public key or its IP address. As a result of this random assignment of IDs, with high probability, nodes with adjacent IDs are diverse in geographic distribution, ownership, jurisdiction, network attachment, etc.

Each node in Pastry maintains a routing state with a routing table, a neighborhood set and a leaf set. Each entry in the routing table of a node x contains the IP address of one of potentially many nodes whose IDs have the same prefix as x. The neighborhood set M contains the IDs and IP addresses of the $|M|$ nodes[2] that are closest (according the proximity metric) to the local node.

[2] $|M|$ is the number of elements in the set of nodes M.

The neighborhood set is not normally used in the routing operation; it is useful in maintaining locality properties. The leaf set L is the set of nodes with the $|L|/2$ numerically closest larger IDs[3], and the $|L|/2$ nodes with numerically closest smaller IDs, relative to the present node's ID. The leaf set is used during the message routing. Typical values of $|L|$ and $|M|$ are 2^b or $2 * 2^b$.

Routing in Pastry is based on the algorithm first proposed by Plaxton et al. in [93], where each node is the root of a tree. The leaves of the tree are the providers of an object replicated at the root and all other nodes of the tree except the leaves, and the root store a pointer to the root nodes. Routing between any two neighboring nodes of the tree is based on the similarities of their prefixes.

To lookup a key in Pastry, messages are routed to the node whose ID is numerically closest to the given key. This is accomplished as follows: in each routing step, a node normally forwards the message to a node whose ID shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's ID. If no such node is known, the message is forwarded to a node whose ID shares a prefix with the key as long as the one of the current node, but which is numerically closer to the key than the present node's ID. The choice of the parameter b involves a trade-off between the size of the populated portion of the routing table, approximately $(\Delta - 1) \log_\Delta N$ entries, and the maximum number of hops required to route between any pair of nodes ($\log_\Delta N$); we note that $\Delta = 2^b$ for clarity reasons. With a value of $b = 4$ and 106 nodes, a routing table contains on average 75 entries and the expected number of routing hops is 5, whilst in a network with 109 nodes, the routing table contains on average 105 entries, and the expected number of routing hops is 7.

Pastry uses the parameter Δ to tune the routing table when necessary and to support better resilience to churn. However, this is only possible at the expense

[3] $|L|$ is the number of elements in the set of nodes L.

of the size of the routing table that expands logarithmically when the number of nodes linearly increases in the system. Besides, Pastry uses a greedy approach of resource lookup that can not ensure that a requested resource is found in the shortest possible path.

Just as Chord and Pastry studied above, the Symphony protocol [83] also presents an approach of self-organization in P2P that is based on a ring distribution model and DHT. It extends the Kleinberg's approach of a Small World construction [72], and achieves a routing table that is of constant size. However, this is achieved at the expense of a larger lookup latency during resource lookup, which is estimated to $\log^2 N$ overlay hops.

3.2.3 Tapestry

The routing algorithm in Tapestry [126] is similar to that of Pastry. Routing is achieved using the prefix-based Plaxton-like approach. Tapestry uses DHT for objects' distribution in the network. Objects in Tapestry can be assigned multiple global unique identifiers (GUIDs), such that objects are published in more than one root node in the network. This is important to support resilience, when a root node accidentally fails.

To locate an object O in the network, a node routes a message to a root node O_R of the object. Each node on the path checks whether it has a location mapping for O. If so, it redirects the request message to the provider of O. Otherwise, it forwards the message onwards to the root node O_R. The authors of Tapestry argue that the resource lookup in a Tapestry-based network with N nodes can be achieved within approximately $\log_\Delta N$ overlay hops, where each node uses a Δ-base identifier as node's identifier (ID).

To support fault-tolerance, when the prefix-based routing scheme can not

be used at a given step, — e.g because the expected node is not present in the "neighbor maps" ("finger table" in Chord),— Tapestry uses an alternative routing approach called "surrogate routing" that routes around the failure region. Each non-existent node's ID in the "neighbor map" is mapped to some an existing nodes with a similar identifier in the surrogate routing solution. The "neighbor map" has a size of $O(\log N)$ entries just like in Pastry and Chord.

Tapestry, however, shows some severe drawbacks with regard to the performance. In fact, each server for a given object in the network periodically advertises or publishes the object's information by routing a publish message toward the root node, increasing the cost of control messages in the system. Besides, nodes use periodic beacons to the destinations in their "neighbor map" to confirm outgoing links and detect node's failures which is an expensive maintenance approach.

Nevertheless, global scale Internet storage applications such as OceanStore [75] and multicast distribution systems such as Bayeux [128] are developed on top of Tapestry. Other applications such as Scribe [101, 32] and Past [45, 100] for distributed web-cache management, data storage and event notification are also based on Tapestry.

3.2.4 Content Addressable Network (CAN)

The Content Addressable Network [94] is a distributed decentralized P2P infrastructure that represents its nodes on a Δ-dimensional hypercube, where each node holds a region of the hypercube. For simplicity reason, the description of CAN in [94] focuses on the description on a 2-dimensional hypercube represented in a Cartesian coordinate system as shown in Figure 3.5.

This coordinate space, — which is only virtual,— is used to store the tuples

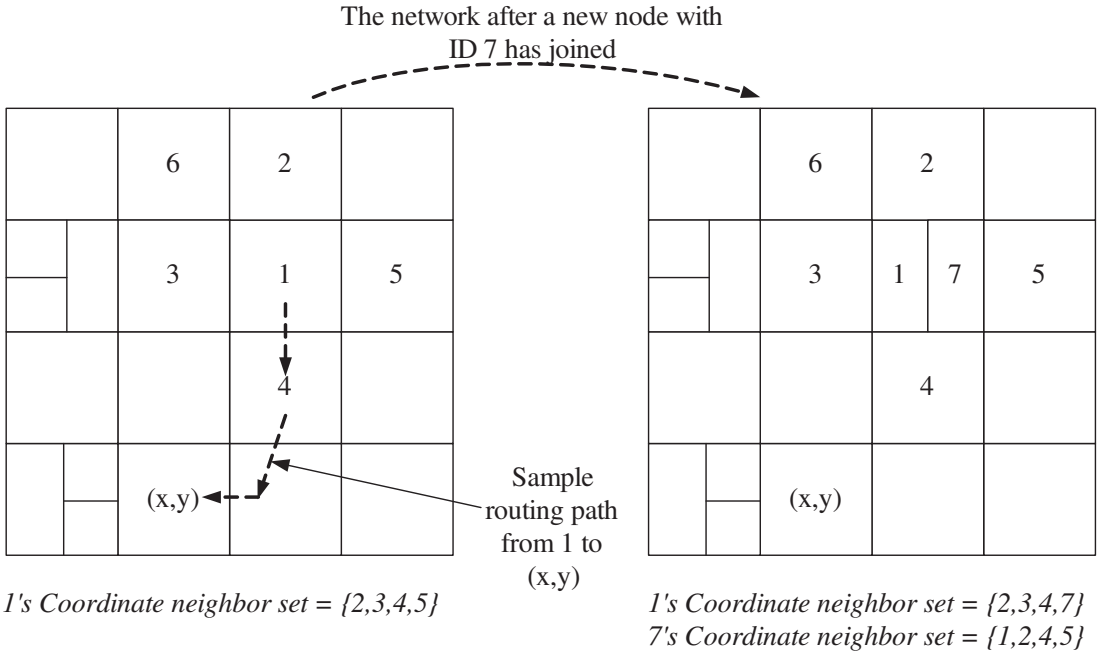

Figure 3.5: An 2-dimensional hypercube for the representation of a CAN overlay network.

(*key*, *value*). To store a tuple (K_1, V_1), the key K_1 is deterministically mapped onto a point P in the coordinate space using a uniform hash function. The corresponding (*key*, *value*) tuple is stored at the node that owns the zone within which the point P lies.

To retrieve an entry corresponding to a key K_1, any node can apply the same deterministic hash function to map K_1 onto point P and then retrieve the corresponding value from the point P. If the point P is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in whose zone P lies. A node learns and maintains the IP addresses of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbors in the coordinate space serves as a coordinate routing table that enables routing between arbitrary points in this space. For example, in Figure 3.5, the node 5 is a neighbor of the node 1 because its coordinate zone overlaps with 1's zone along the Y-axis and abuts

along the X-axis. On the other hand, node 6 is not a neighbor of 1 because their coordinate zones abut along both the X- and Y-axes. A CAN message includes the destination coordinates, which describe a point of a region own by the destination CAN node. Using its neighbor coordinate set, a node routes a message toward its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination as depicted in Figure 3.5.

A new peer joining the system must have its own portion of the coordinate space allocated. This can be achieved by splitting existing peer's zone in half; retaining half for the peer and allocating the other half to the new peer; the join process is depicted in Figure 3.5 where 7 joins the network by splitting the zone owning by 1 into two contingent zones sharing the set of nodes $\{2, 4, 5\}$ as common neighbors. CAN has an associated DNS-like domain name which is resolved into IP address of one or more CAN bootstrap peers (which maintain a partial list of CAN peers), when a node is joining the network.

When a peer leaves the CAN network, an immediate takeover algorithm ensures that the region of the failed peer is taken over by another peer until the failure is repaired. The peer updates its set of neighbors to eliminate those peers that are no longer its neighbors. Each peer in the region of the hypercube affected by the change sends soft-state updates to ensure that all of their neighbors will learn about the change and update their own neighbors' set. The number of neighbors a peer maintains is dependent only on the dimensionality of the coordinate space which is independent of the total number of peers in the system.

CAN supports fault-tolerant routing by forwarding a message to another node in the neighbor coordinate set that is closer to the destination than the node itself. In term of performance, CAN can achieve a different performance profile than the other algorithms such as Chord, Pastry and Tapestry. State information kept locally at each node is constant $O(\Delta)$ and the maximal path length has a

diameter $D = \Delta \sqrt[\Delta]{N}$ overlay hops.

Although the Δ-dimensional topology presented in CAN is intuitive and easy to understand, the robustness of the approach is not mature enough for usual failure scenarios. For example, CAN does not handle the case of ungracefully departure. In addition, routing is based on greedy forwarding that does not necessarily ensure a shortest path to the requested resource.

Other content addressable related research results have been presented in [46, 104] to deal with network resilience in case that a probabilistic set of nodes concurrently join or ungracefully leave the network. Along with CAN, some other protocols which rely on the hypercube concept have been proposed, e.g. HyperCup [107] and Ulysses [76]. The network construction approach of the HyperCup approach builds the overlay gradually from a Δ-dimensional to $\Delta + x$-dimensional overlay when new nodes join the network. HyperCup achieves message routing with $\log N$ overlay hops and supports resilience using a CAN similar approach. Thus, it shares the above limitations of the CAN protocol.

3.2.5 Shufflenet-based P2P Systems

Shufflenet has been used in the past in other research domains to design fault-tolerant systems [109, 30]. The shufflenet is also known as butterfly network and has recently gained more and more interest for the interaction design of self-organized systems such as P2P systems.

Generally, a D-dimensional butterfly graph of degree Δ is a directed graph denoted as $BF(\Delta, D)$ whose vertices are tuples (w, l), where w is a Δ-base string of length D and l is an integer in the interval $[0,\ D]$. A directed edge from vertex (w, l) to $(w', l+1)$ exists if and only if $w' = \Delta * w + k$ with $k \in [0, \Delta[$, for instance there are directed edges from the node (100, 0) to the nodes (001, 1) and (000, 1)

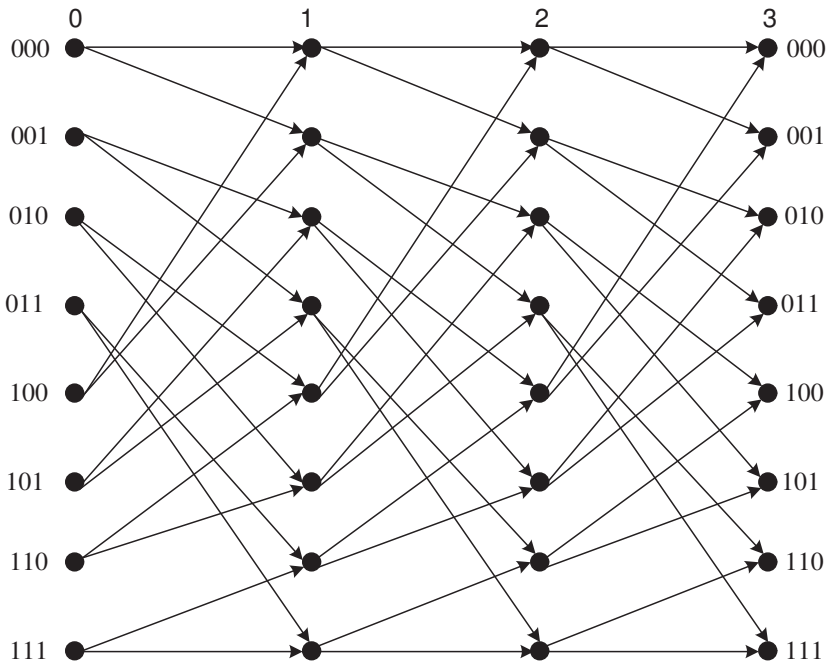

Figure 3.6: An example of a $BF(2,3)$ butterfly network with 32 nodes.

because,

$$100 * \Delta + k = \begin{cases} 000 & for \quad k = 0 \\ 001 & for \quad k = 1 \end{cases} \tag{3.1}$$

for $\Delta = 2$ as shown in Figure 3.6.

A D-dimensional butterfly graph has totally $(D+1)\Delta^D$ vertices and $D\Delta^{D+1}$ edges. Figure 3.6 shows an example of a $BF(2,3)$-Butterfly network with $N = 4 * 2^3 = 32$ nodes. Butterfly was designed for management of interconnection between the nodes in a static network. Viceroy [82] and Ulysses [76] are both non-hierarchical approaches of P2P systems which are based on the butterfly network concept. They propose an adaptation of the butterfly to dynamic systems such as P2P systems. In term of performance, they can achieve object lookup within a maximum of $\log N$ routing hops while maintaining a constant routing table size. However, the management of the hypercube-like overlay is very complex

in Uylsses. Viceroy does not directly address the issue of fault-tolerance during network churns.

3.2.6 De Bruijn

The studies performed in [80] have shown that De Bruijn digraphs [41, 43] can significantly improve the network's diameter over all other approaches such as Chord, CAN, Pastry, Tapestry and Viceroy. A $B(\Delta, D)$ De Bruijn graph defines a fixed-degree Δ network with an optimal diameter $D = \log_{\Delta} N$, for a given number of nodes in the network of size N. The choice of Δ is a significant performance parameter and thus, it can be varied to improve reliability. In fact, Imase and Itoh have constructed in [62] a $B(\Delta, D)$ nearly optimal De Bruijn digraph, where the number of nodes $N = \Delta^D$ is relatively close to the Moore bound for a large value of Δ.

However De Bruijn graphs were primarily conceived for interconnection between nodes in a static environment with a constant number of nodes $N = \Delta^D$. Thus, to construct and maintain a De Bruijn-based network is a challenging task, when some nodes can join continually and some others can leave unpredictably. Hence, the dynamic nature of P2Ps limits the efficient applicability of the De Bruijn digraph concept. For instance, the network capacity N in a dynamic network varies and $N = \Delta^D$ is not always valid. As a result, routing in at most $D = \log_{\Delta} N$ hops can not always be taken for granted.

Despite this conceptual limitation of primary De Bruijn graph designed for interconnection in static networks, it presents some interesting properties such as the constant degree of its nodes, the optimal network diameter and the network connectivity. A family of P2P overlays based on De Bruijn graph has recently emerged such as ODRI [80], CAN-D2B [47], distance halving [87], Broose [48],

Koorde [68] and Pagoda [27]. They try with more or less success to adapt the De Bruijn static topology to a dynamic environment like P2Ps. Although they can achieve some ameliorations of the P2P performance, they demonstrate some fundamental limitations. The Koorde protocol for instance assumes a maximal number of nodes in the P2P system. This is a non-flexible approach because one can not increase the size of the system at runtime over a given limit. Koorde, Broose [48] and Pagoda [27] achieve a lookup performance, which is not better than the original De Bruijn solution although with low routing state per node.

3.2.7 Other Approaches of Flat DHT P2Ps

Some other constant-degree P2P architectures such as Cycloid [108] and Hypercup [107] have been proposed recently to improve the lookup performance of request in large-scale and highly dynamic systems. Cycloid emulates a Cube-Connected-Cycles (CCC) graph that can achieve a time complexity of $O(\Delta)$ per lookup request by using 7 neighbors per node, where $N = \Delta 2^{\Delta}$. The simulation results in [49] show that Cycloid performs better than Koorde and Viceroy in large-scale and dynamic P2P systems. However, the Cycloid proposal does not handle the aspect of ungracefully departure from the network.

In our recherche however, we could not find any proposal of structured P2P architecture that significantly improves the resource lookup path length below the $\log_{\Delta} N$ bound with a routing table that is lower than $\log N$. Interesting approaches such as Koorde [68] and D2B [47] are mostly extremely complex and uses many non-valid assumptions to address fault-tolerance, while maintaining high performance.

P2P Approach	**Overlay & Next hop**	**Path Length**	**Node's state**	**Update**	**Join or Leave**
Chord	ring & the node of the finger table with closest ID to the destination	$\log_2 N$	$O(\log N)$	periodic $O(\log N)$	$O(\log N)$
Pastry	ring & the node of the routing table with matching prefix	$\log_\Delta N$	$2\Delta \log_\Delta N$	$O(\Delta \log_\Delta N)$	$\log_\Delta N$
Tapestry	tree structure & the node of the routing table with matching prefix $\Delta = 2^b$	$\log_\Delta N$	$\Delta \log_\Delta N$	Periodic $O(\Delta \log_\Delta N)$	$O(\log_\Delta N)$
CAN	Δ-dimensional hypercube & the node with the closest ID to the destination	$\Delta \sqrt[\Delta]{N}$	Δ	Δ	Δ

Table 3.1: A summary of the performance parameters for strictly structured flat DHT-based P2P architectures.

3.2.8 Summary of Strictly Structured P2Ps

This subsection presents the summary of the comparative study for flat DHT-based P2P systems with strict structures. Table 3.1 highlights the overlay design and the next hop selection criteria. We show in this table that the node's state or the routing table size as well as the control overhead disseminated during the node joining or leaving phase can be constant or logarithmic depending on the technique. Basically, all the investigated protocols have been designed for supporting fault-tolerant operations.

3.3 Loosely Structured P2Ps

In loosely structured P2P systems, the overlay structure is not strictly specified. The construction of the system does not impose any structure. Loosely structured P2P systems can be considered as nearly unstructured because the overlay is based on some partial order instead of relying on total randomness (as in pure unstructured). It is either formed based on hints, — e.g. *how good is the chance to get a result through a given neighbor?*),— or formed probabilistically (random neighbor selection). In Freenet [36] and Phenix [119], the overlay evolves into the intended structure based on hints or preferences. In Symphony [83] and in the small-world-based model of Freenet proposed in [125], the overlay is constructed probabilistically. In Freenet, data is stored based on the hints used for the overlay construction. Therefore, searching in Freenet is also based on hints. In Phenix [119], the overlay is constructed independently of the application. The data location is determined by the applications. Therefore searching in Phenix is application-dependent. In Symphony [83], the data location is clearly specified by using DHT, but the neighborhood relationship is probabilistically defined. Searching in Symphony is guided by reducing the numerical distance from the querying source to the node that stores the desired data.

Loosely structured P2Ps can be subdivided into two significant groups of power-law overlays and small-world model overlays. These two groups are studied in the following two sections.

3.3.1 The Power-law Graph Overlay

A graph formed by the network which degrees follow a power-law distribution is called a power-law graph. This means that the probability to find a node with

the degree k is

$$P_k = ck^{-\gamma}$$

where γ is a positive integer and c is a constant. Power-law signature has been observed in natural complex systems such as social networks and ecological networks, as well as man-made complex systems such as the Internet and the World Wide Web with a value of γ close to 2. Freenet [36] and Gnutella [8] tend to evolve into a power-law graph with a power-law exponent close to 2. A review of efficient strategies for searching power-law overlays is proposed in [16]. An approach has been presented in [16] that proposes to using high-degree nodes as dominant nodes for efficient routing and resilience. Thereafter, a node looking for an object will forward the query to its neighbor with the highest degree. The query is subsequently routed to the neighbor with the next highest degree until the provider is found.

Phenix [119] makes a power-law overlay "organically" emerge by guiding the node's join process through preferences. Phenix has been derived from popular unstructured P2P network architectures such as Gnutella [8] and KaZaA [5]. In Phenix, the new nodes will prefer to connect to existing nodes with high degree. Specifically, when a new node joins the system, it gets a list of live nodes using a rendezvous mechanism. This list of nodes is divided by the joining node into two sets of random and friend neighbors. Next, the new node sends a ping message with TTL = 1 to each friend, who returns its own neighbors in a pong message to the new node. Each friend also forwards the ping message to its own neighbors. On receiving the ping message, the friend's neighbors add the new node to their own special lists. The new node's friends and the neighbors of the friend form a candidate list of the new node. The list of candidates is then sorted based on the decreasing order of the number of the node's appearance. The top c number of

nodes in this list is selected as the preferred list of the new node. The new node then creates connections to all nodes in its random list and preferred list. The new node also periodically contacts the neighbors in its preferred list for possible backward connections. A preferred neighbor increases its counter each time it is contacted by the new node. If the counter reaches a constant value r, the preferred neighbor decreases the counter by r and adds a backward connection to the new node. The node maintenance procedure checks periodically the number of neighbors of every node, and if this number becomes lower than a specific limit, a procedure runs and creates proper neighbors for this node.

3.3.2 The Small-world Model Overlay

A small-world graph is a graph in which each node has many local connections and a few random long-range connections. The small-world model for P2P originates from the small-world phenomenon (also known as the small-world effect). It is an interesting starting point for researching interconnection in self-managed dynamic distributed systems. The small-world model is based on the hypothesis that everyone in the world can be reached through a short chain of social acquaintances. According to a 1967 small-world experiment performed by psychologist Stanley Milgram, the concept coined to the famous phrase "*six degrees of separation*". Milgram found that two random US citizens were connected by an average of six acquaintances. However, after almost forty years now its status as a description of heterogeneous social networks still remains an open question. The diameter of a small-world graph is $\log^2 N$ [72].

Symphony [83] employs the small-world graph model to implement the DHT on a circular overlay. A data item with a key k is mapped onto a node whose ID is numerically closest to k. Each node has two short-distance links to its

immediate predecessor and successor in the ID ring and m long-distance links to other distant nodes in the ring. These distant nodes are chosen probabilistically. Specifically, when selecting a long-distance neighbor, a node A draws a random number r based on the probabilistic distribution function: $P_N(r) = \frac{1}{r \log N}$, where r is in $[1/N, 1]$ and N is the current number of nodes in the P2P. Then, the node A finds another node B that is responsible for the number r. This node B is selected as one long-distance neighbor of node A. The parameter m is determined experimentally. N is estimated based on the sum of segment lengths managed by a set of distinct nodes. Symphony has a unidirectional and a bidirectional routing protocol. In the unidirectional approach, each node on the query path forwards the query to one of its immediate or distant neighbors that is clockwise closest to the sought key. In the bidirectional routing protocol, the query is forwarded to one immediate or distant neighbor that has the absolutely shortest distance counterclockwise from the responsible node. The 'lookahead' approach is proposed to reduce the query latency even further. In this approach, each node looks ahead at its neighbors' neighbors during query forwarding. For example, in the 1-lookahead approach, each node forwards a query to the neighbor whose neighbor is closest to the sought key.

In Freenet, the uniform cache replacement policy 'least recently used' (LRU) for data-stores management can destroy the key clustering and the resource-awareness in both the data-stores and the routing tables when the number of data stored is large. The work in [125] solves this problem by incorporating a small-world model in the data-store cache replacement policy. Routing tables are tailored by data-stores. Integrating a small-world model in the data-store cache replacement policy makes the routing tables emulate a small-world model overlay. Specifically, each new node selects a random seed from the key space. When a new data item with a new key k comes in and the data-store is full, the

node first compares the new key with the existing key k' in its data-store that is farthest from its seed. If the new key k is closer to the seed than k', then the node removes the data with key k', stores the new data with key k, and adds a new entry for k in the routing table. (This is intended for emulating short links to close neighbors in a small-world model.) Otherwise, the node probabilistically removes k', caches k, and adds a new routing table entry for the new key k. (This is designed for emulating a small number of random long links to distant neighbors in a small-world model). The above scheme enforces clustering of keys around the random seed at each node's data-store and routing table.

3.4 Summarizing Discussions and Conclusion

Having investigated different classes of P2P systems, we come to the conclusion that there is no viable solution for resource location so far. The recently proposed solutions are not optimal for resource management in P2P systems according to the requirements of Section 1.2 in Chapter 1. Each class of the investigated solutions presents strengths and drawbacks.

Unstructured P2P systems do generally not maintain any structure and therefore do not need any costly algorithm for network maintenance and links update. But, exactly because it requires no predefined structure, the resource lookup operation is mostly non-deterministic and is achieved by techniques based on flooding and its derivates. Besides, the resource lookup operation does not guarantee any success even when the requested resource exists in the network. Moreover, the degree-diameter relationship is mostly not optimal with a lookup path length varying between 1 and $N-1$ hops for a network of N nodes and each node maintaining information about any other nodes in the network. In the worst case, when a node is requesting a rare resource, almost all the nodes in the system can

be involved in the lookup operation.

Strictly structured P2P systems have been proposed to overcome the limitations of unstructured architectures. A large number of strictly structured P2P architectures have been based on distributed hash table. DHT is used to deterministically mapped keys to nodes in the network. DHT enables exact matching of queries in P2P networks. The lookup operation involves only a part of the network and the resource can always be found with high probability when it exists and when it is described with a fine precision (using hashing for example). The DHT concept ensures an evenly balanced distribution of keys to nodes in the network, with an average load of $\frac{N}{K}$ pro node in an N-nodes network with K total number of keys. However, their strictly defined structure also requires a strict maintenance strategy. To keep the network in a consistent state, the cost for network maintenance is generally large with a complexity order of $O(\log N)$ control messages per node. The routing strategy in many studied cases is based on a greedy forwarding mechanism, which is indeed locally optimized, but does not always choose the shortest path from source to destination. Another limitation of strictly structured P2P architectures is their overlay network which does not necessarily match with the physical topology of the network. To enable an easy correlation between physical topology and overlay construction, hierarchical approach of DHT-based P2P systems have been proposed. By introducing the notion of hierarchy into the overlay construction method, the locality-awareness property could be achieved at the expense of load balancing. In fact, the use of hierarchy in DHT-based P2P architectures destroys the load balancing property of DHT in the overall network formed by the different hierarchy clusters or groups.

Since DHT is designed to support only exact matching of queries, the non

DHT-based approaches of strictly structured P2P architectures have been proposed to support range queries as well. However, they rely on strong replication of network resources to ensure a broadly awareness of network elements. With strong replication, a requested resource can be found even when it is only approximately described.

Loosely structured P2P architectures aim at simultaneously addressing the limitations of unstructured and strictly structured P2Ps. They are mainly designed to support low-cost network maintenance and high availability. These architectures are based either on balanced trees or on skip list concepts. However, these approaches also rely on strong replication of resources on each node to achieve resource lookup with length of $\log N$ which is still far from the Moore Bound with a local routing state in the complexity order of $O(\log N)$.

The investigation in this overview reveals both DHT and the De Bruijn graph as interesting approaches for load balanced resource distribution and efficient routing in a P2P overlay structure. The De Bruijn digraph enables a lowest nearly optimal bound of $\log_{\Delta} N$ overlay hops during lookup operation, which is closest to the Moore Bound for a constant routing table size Δ. We argue that the usage of the both approaches, (i) the adaption of De Bruijn graph to support dynamic network, and (ii) the extension of DHT to support range queries and low-cost network management, could provide a good and adequate background for an effective P2P solution. Hence, the results extracted from this investigation constitute a foundation for the development presented in the next chapters of this thesis.

Chapter 4

A Concentric Multi-ring Overlay for P2P Systems

IN large-scale dynamic distributed systems such as P2P systems, the number of nodes forming the network can grow to several thousands, millions or even billions. These systems are designed and maintained based on the concept of self-management of nodes, which can join at anytime or leave without any prior warning. As we have presented in the precedent Chapter 1, the issue of efficient resource location in a P2P system is not trivial, because it can not be assumed that all nodes are aware of all the resources in the system at a given time. The current chapter describes an original approach to the management of P2P systems that enables fast and flexible resource distribution and location.

Here, we present a construction mechanism for a virtual network topology, — the concentric multi-ring (CMR) overlay,— to be deployed upon a network infrastructure such as the Internet. The idea is to organize the resources of the network such that they can be found independently from the routing protocol of the underlying IP network. This virtual network that lays over the routing level of an initial network is generally called P2P overlay in the P2P jargon.

The P2P overlay presented in this chapter virtually organizes the resources of a P2P network in a concentric multi-ring architecture. The CMR architecture enables an efficient organization of a large number of resources in such a way that each lookup operation can be completed successfully along the shortest path. The resource distribution strategy is based on a slightly modified approach of the distributed hash tables (DHT) enabling both exact and range matching of queries. In fact, each resource of the network is replicated in a number k of nodes, with $k \in [1, D_{max}]$ and D_{max} being the number of rings formed by the CMR architecture. The message routing between the nodes is realized using the De Bruijn neighborhood concept on each ring of the CMR overlay. In what follows, we give a detailed description of the overlay construction strategy, as well as a detailed description of the resource distribution, the message routing and the object lookup schemes.

4.1 Notations and Definitions

Before starting, we will introduce in this section some notations and definitions that may help along to better understand the rest of this chapter.

Each node on the concentric multi-ring overlay has one identifier denoted as "ID". The ID is expressed as a Δ-base integer formed of D digits, where D is the diameter and Δ is the degree of the De Bruijn digraph[1] formed by the ring whereon the node is located. In the following description and throughout the document, a node x is represented by its ID, encoded as $x = x_{D-1}x_{D-2}x_{D-3}...x_1x_0$. For example, a node with ID $x = 020102$ is a node located on the sixth concentric ring, with $\Delta = 3$.

[1]See definition of De Bruijn digraph in Section 4.2.

Further, each expression used in a mathematical operation should be considered as a Δ-base integer. In this document, we use the terms "resource" and "object" interchangeably to express a piece of file, memory, processing power, bandwidth or simply a resource. The resource provided by a node x is denoted as R, where R is a Δ-base integer known as the global unique identifier (abbreviated GUIDE).

4.2 Background: De Bruijn Digraph

The objective in this work is to design a cost-effective and lookup efficient resource management solution for P2P systems according to the P2P network characteristics such as decentralization, dynamic and distribution. Thereafter, the network should be organized such that:

- ◇ any new node joining the system is inserted by involving only a small subset of the nodes in the network. This is important to avoid that a request is disseminated to the whole network each time a node joins the system.
- ◇ any node leaving the system (voluntary or ungracefully) should have only the smallest possible impact on the interconnection of other nodes. This is important to resist to churns and ensure a high degree of network stability.
- ◇ the network diameter is as close as possible to the Moore bound, even when some nodes join and leave continually.

A De Bruijn digraph [41] presents several interesting properties that can be exploited to facilitating the resource discovery in P2P systems. In what follows, we give a definition of the De Bruijn digraph along with an analysis of its properties.

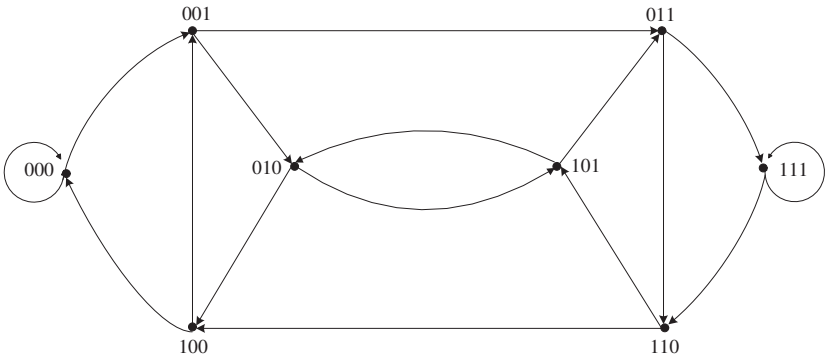

Figure 4.1: Example of De Bruijn digraph

4.2.1 Definition of De Bruijn Digraph

A De Bruijn digraph, denoted as $B(\Delta, D)$, with degree Δ and diameter D, is the directed graph such that:

$$\begin{aligned} Z_\Delta &\stackrel{déf}{=} \{0, 1, ..., \Delta - 1\} \\ Z_\Delta^D &\stackrel{déf}{=} \{x = x_{D-1}x_{D-2}...x_1x_0 \,|\, x_i \in Z_\Delta\} \\ \Gamma^+(x) &\stackrel{déf}{=} \{(\Delta * x + i)\, mod\, N,\ i = 0, 1, ..., \Delta - 1\} \end{aligned}$$

where "*mod*" is the modulo function, Z_Δ^D is the set of nodes x forming the $B(\Delta, D)$-De Bruijn digraph and $\Gamma^+(x)$ is the set of neighbors for each node x.

Per definition, a $B(\Delta, D)$ De Bruijn graph forms a network of size $N = \Delta^D$. Figure 4.1 illustrates an example of a $B(2, 3)$ static De Bruijn network with 8 nodes.

4.2.2 Why Using De Bruijn?

The analysis of the De Bruijn graphs in related work [41, 43, 62, 65, 64] have shown interesting properties for their applications to static interconnected networks:

Constant degree: each node in the network has a constant number of Δ incoming and Δ outgoing edges. This means, the routing table size of nodes in the network is constant and not dependent on the number of nodes in the system.

Optimal diameter: in an N-node network, there exists a path of length $D = \log_{\Delta} N$ hops between any two nodes in the network.

Connectivity: studies on the connectivity of the De Bruijn networks have shown that at most $\Delta - 1$ nodes can fail without disconnecting any node or any set of nodes from the network. Hence, the degree Δ is an important design parameter that can be varied to improve connectivity.

Moreover, the De Bruijn digraph concept is a mature one. In fact, there are related researches on using De Bruijn graphs that have been carried out in the area of VLSI/ULSI design or multiprocessors interconnection since the early *90ies* for solving the problems of fault-tolerant routing and flexible interconnection [34, 105].

4.2.3 Routing in a De Bruijn Network

The lookup operation of a resource R in a De Bruijn network is equivalent to the routing operation from a node A to a node $B = \mathit{prefix}(R)$, which eventually knows a provider of R. The pseudo-function $\mathit{prefix}(R)$ returns the first k elements most left from the Δ-base expression of R, with $k \leq D$. The routing operation is achieved using a consecutive left shifting of A with R until the node B is reached. The left shifting operation on the node A results in a node

$$B = (\Delta * A + i) \, mod \, N, \;\; i \; = \mathit{topLetter}(R, k) \tag{4.1}$$

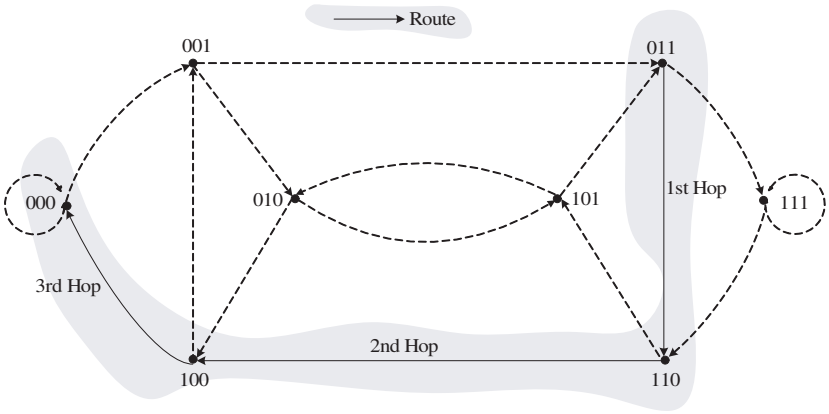

Figure 4.2: An example of message routing from the node $x = 011$ to the node $y = 000$

where $topLetter(R, k)$ is a function that returns the k^{th} letter from left of R and k is the covered distance from the source. At any node A, a next-hop node B is chosen after left shifting of A such that B is a neighbor of A. Thus, routing in this manner in the De Bruijn graph needs at most $D = \log_{\Delta} N$ overlay hops from source to destination. Taking the example in Figure 4.2, a message to be routed from 011 to 000 for $\Delta = 2$ will take the highlighted path

$$011 \xrightarrow{0} 110 \xrightarrow{0} 100 \xrightarrow{0} 000$$

. According to the equation 4.1, the message will be forwarded from 011 to $(2 * 011 + 0)\, mod\, 8 = 110$ with $i = 0$ in the first hop. In the second hop, the message will be forwarded from 110 to $(2 * 110 + 0)\, mod\, 8 = 100$. In the third hop, the message will be forwarded from 100 to $(2 * 100 + 0)\, mod\, 8 = 000$.

4.2.3.1 Limitations of De Bruijn

A De Bruijn digraph looses its interesting design properties when more than $\Delta - 1$ nodes leave the network simultaneously. And, dynamic distributed systems such as P2P systems are characterized by their strong dynamic behavior; nodes can

leave individually or collectively without any prior warning. In fact, the De Bruijn digraph was conceived for nodes organization in static environments with the assumption that all the N nodes are always available in the system. When one node fails, the network diameter $D = \log_{\Delta} N$ can not be longer guaranteed; although a De Bruijn digraph is connected even when at most $\Delta - 1$ nodes fail from the network.

Another limitation of the De Bruijn digraph concept is its poor degree of extension. De Bruijn graphs are designed with a fix structure that can not easily be modified to support a larger number of nodes than originally planned. For instance, if a $B(\Delta, D)$ De Bruijn digraph has to be modified offline or at runtime to support more than $N = \Delta^D$ nodes, the next possible consistent De Bruijn digraph that can be built should have $N = \Delta^{D+1}$ or $N = (\Delta + 1)^D$ nodes. In any of the two cases the identifiers of the nodes should be modified as well as the links between them. These are why De Bruijn graph can not be efficiently applied in its basic version to designing fault-tolerant systems such as dependable and reconfigurable computing systems.

However, the way we use the De Bruijn digraph concept in this work eliminates these shortcomings. We propose an intuitive and flexible design approach based on rings and an adaptive De Bruijn-based routing solution, which are presented in more detail in the following sections.

4.3 CMR: System Overview

As the name already reveals, the concentric multi-ring overlay arranges the nodes in a set of concentric rings as shown in Figure 4.3. Each ring is built up of geographically distributed nodes, which form a De Bruijn network. Two neighboring nodes x and y on a ring can be physically wide from each other. It is possible

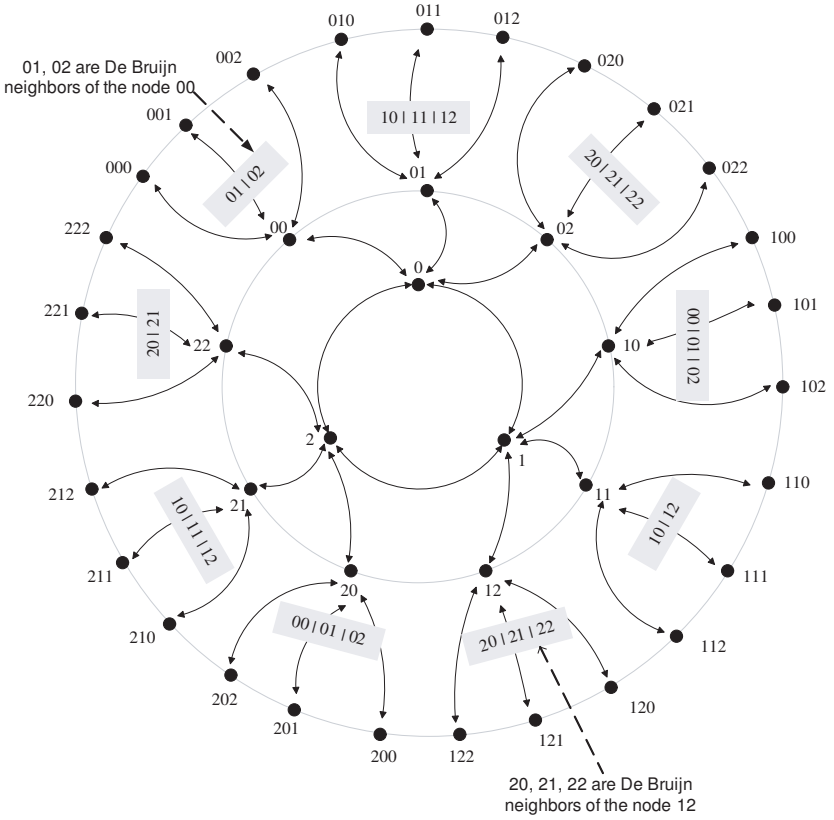

Figure 4.3: System design: Concentric Multi-ring overlay with $\Delta = 3$

that a node x which is located in Berlin is logically neighbor of a node y located in Boston. A ring is identified by a ring's ID. The ring's ID is the diameter D of the De Bruijn digraph formed by this ring. Each ring with diameter D has Δ^D nodes.

The concentric multi-ring overlay is so constructed that the ring with the smallest radius is the innermost ring with ring's identifier equal to $D = 1$, the ring with the next larger radius is the ring with $D = 2$ and so forth.

When a new node joins the network, it is assigned a new unique node's identifier on the outermost ring with ID D_{max}. That is, the new node is assigned an identifier of length D_{max}. The identifier of a node x on the ring with diameter D is denoted as $x = x_{D-1}x_{D-2}...x_1x_0$. Figure 4.4 gives an overview on how the overlay is gradually formed when the network is growing from one to 16 nodes.

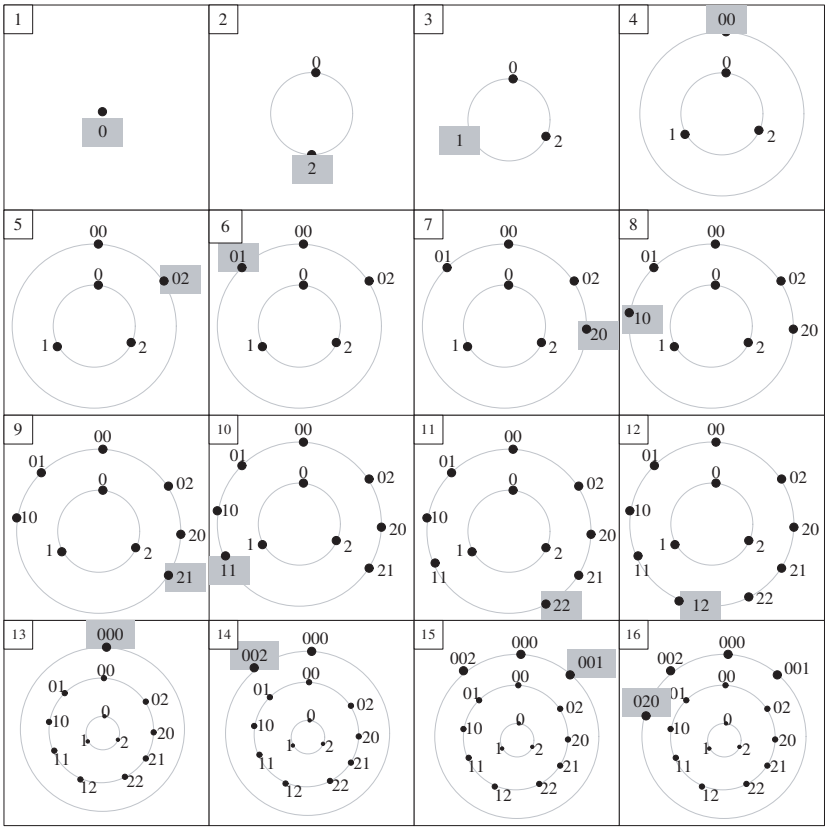

Figure 4.4: Gradually construction of the overlay for $\Delta = 3$

Because the network topology changes continually with nodes joining or leaving the system, we have defined a set of rules that are used to construct the overlay such that the image of the network is always in a consistent state. These rules should not be considered as invariant or assumptions but as construction laws. In order to exploit the routing properties of the De Bruijn digraphs, the first rule stipulated that:

Construction rule 1 *The nodes on an inner ring with $D \leq D_{max}$ are so connected that they always form a $B(\Delta, D)$ De Bruijn network.*

Application of the Construction rule 1 ensures that the interconnection between any two nodes on an inner ring is achieved in at most $\log_{\Delta} N'$, where $N' < N$ is the total number of nodes on the ring.

The logical multi-ring representation is conceived with the idea that the nodes on the inner rings are more reliable than nodes on the outer ones during the gradual network expansion. Moreover, the concentric multi-ring topology can also allow routing in multiple directions[2], which improves flexibility and reliability in the case of nodes' failures. However, since nodes may also fail in the inner rings during the system evolution, the overlay construction strategy should ensure that the inner rings are always in a consistent state; thus, the following rule must always be applied:

Construction rule 2 *When a node leaves (ungracefully or voluntary), a stabilization routine should be started on a finite, possible small portion of the network. During the stabilization procedure some nodes may change their logical ring and hence their status in the concentric multi-ring topology.*

The question how the churn situations are managed to restore network consistency is discussed thoroughly in the next chapter. The following construction rule defines an important expansion operation of the CMR network.

Construction rule 3 *When a new node joins the network and the outermost ring with identifier D already has Δ^D nodes, a new ring with identifier $D+1$ should be created to locate the new node as shown in Figure 4.4.*

With the Construction rule 3, the network can grow as necessary. There is no upper limit for the maximal number of nodes as in Koorde [68] or D2B [47]; hence there is no limited maximum for the number of rings in the network. Moreover, the initialization of a ring should obey to the following construction rule:

[2]multi-dimensional routing: consider a multidimensional $3D$ or $4D$ or xD representation of the model, which adds new surface for trajectories along the routing path. In the sequel, we consider only "flat" (2 dimensional: $2D$) multi-ring topologies for simplicity reasons.

Construction rule 4 *The construction of a ring begins always with the node* $x_{\{init,D\}}$*, which has the lowest identifier on the ring. A node* $x_{\{init,D\}}$ *is present as long as the ring with identifier* D *exists. With* $x_{\{init,D\}} = 00...000$*, where* $1 \leq |x_{\{init,D\}}| \leq D$*.*

The node $x_{\{init,D\}}$ plays the role of a ring contact point during the join operation and ensures a balanced distribution of nodes on the ring, by performing key space halving as described in the next chapter. Each node on the ring can take the responsibility of being a ring contact node, when the node $x_{\{init,D\}}$ fails.

The CMR overlay is constructed progressively as nodes join the network. Each ring is gradually filled and a new ring is created when an underlying one is full. In a CMR overlay constructed with k rings, the first ring consists of exactly Δ^1 nodes, the second ring contains Δ^2 nodes and the k^{th} ring contains Δ^k nodes. Hence, the maximal number of nodes in the system is given by the following lemma.

Lemma 4.3.1 *At a given time, when the CMR overlay is formed of* D *rings, the maximal number of nodes in the network is* $N = \frac{\Delta^{Dmax+1}-\Delta}{\Delta-1}$*.*

Proof: According to the Construction rule 3, a new ring with identifier D is constructed only if the ring with identifier $D-1$ already has Δ^{D-1} nodes. In this way, let D be the identifier of the outermost ring, then the maximum number of nodes in the network is the geometric sum $-1 + \sum_{D_{max}}^{i=0} \Delta^i$ which is at most $N = \frac{\Delta^{Dmax+1}-\Delta}{\Delta-1}$ for $\Delta \geq 2$. ■

In the following sections, we describe in more details how routing, object distribution and location are performed in the CMR network.

4.4 Node's State in CMR

To route and resolve requests toward its destinations in the network, each node in the CMR overlay maintains a routing and a location table.

4.4.1 Routing Table

The routing table is used to choose the next hop toward a given destination. According to the topology of Figure 4.3, a node in CMR maintains a table with at most $2\Delta + 2$ entries. These are the Δ neighbors on the next outer ring, the Δ neighbors on the same ring also called De Bruijn neighbors, the inner ring neighbor and the head node. The head node of a node $x = x_{i-1}x_{i-2}...x_1x_0$ is the node $x^{head} = 0x_{D-1}x_{D-2}...x_1$ or $y = 0x_{D-1}x_{D-2}...x_2$ if x is the ring's contact node.

Figure 4.5 presents an exemplary neighborhood description of the node with ID 100100. This node can route in three different directions: (i) it can route to the three nodes with IDs 001000, 001001 and 001002 on the same ring by left shifting[3] of its own ID, (ii) it can route to the three nodes with IDs 1001000, 1001001 and 1001002 on the next outer ring using the *outwards_forwarding* routine[4], and (iii) it can route to the node with ID 10010 on the next inner ring using the *inwards_forwarding* routine[5]. Hence, using its routing table, a node can forward a message in an ad-hoc manner to one of its neighbors depending only on the identifier of the destination. The neighbor's specification of nodes is given below.

[3]See Section 4.6.1.
[4]See Section 4.6.2
[5]See Section 4.6.3

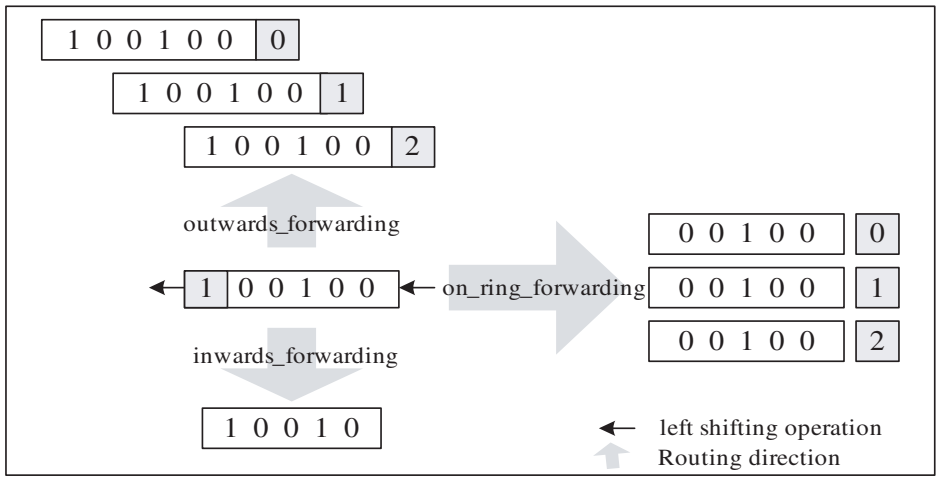

Figure 4.5: Node routing state and forwarding operations for the node 100100.

4.4.1.1 The inner ring neighbors

Each node $x = x_{i-1}x_{i-2}...x_1x_0$, which is not located on the innermost ring, has one inner ring neighbor x^{in} such that $x^{in} = x_{i-1}x_{i-2}...x_1$. The parameter $i = D$ is the identifier of the ring where x is located.

4.4.1.2 The outer ring neighbors

Each node $x = x_{i-1}x_{i-2}...x_1x_0$ has at most Δ outer ring neighbors x^{out} such that $x^{out} = x_{i-1}x_{i-2}...x_1x_0\beta$ with $\beta \in \{0, 1, ..., \Delta - 1\}$. Some outer ring's neighbors may be absent when the next outer ring is not already fully constructed. The parameter $i = D$ is the identifier of the ring where x is located.

4.4.2 Location Table

In the location table, each node maintains a list of pairs (GUIDE, provider IP Address). The resources in the location table are whether the local resources or third party resources for which the node is the ambassador. We define the term "ambassador of a given resource" as a node that is responsible for that resource in the network. There can be many ambassadors for a given resource as described

in the next Section 4.5.

4.5 The Resource Distribution Scheme

In a dynamic distributed environment such as a P2P system, each node joins the network with objects or resources to be shared. In a grid computing environment for example, a grid machine provides processing power or free disk space for data storage. The information about the resources is either kept locally, or distributed to different replicators or ambassadors in the network. When a node needs a resource in the system, it sends a request to a node that is likely to be an ambassador or a replicator of the desired object.

Related approaches use distributed hash tables (DHT) to map resource to node in the network. However, the DHT concept presents the limitation that it assigns the responsibility for one data item to only one node. Besides, it is difficult to lookup for a data that is only approximatively described.

In this work, we use hashing to compute a key or GUIDE for a given data item. However, instead of finding only one ambassador for each data item or key, we choose for each GUIDE at least one and at most D_{max} ambassadors, — D_{max} is the total number of rings of the CMR architecture. For the hashing itself, one can use the well-known SHA-1 function with additional strategies to avoid collision of GUIDEs; the consistent hashing [70] is a good candidate for that purpose. Let us consider that the GUIDE of a resource is of length i for example, then the resource with GUIDE $R = r_{i-1}...r_0$ provided by the node x is a Δ-base integer that is replicated on the nodes on the next inner ring and on the outer rings.

In fact, after joining the CMR network, each node x chooses ambassadors for each of its resources. At a given time, we formally define the ambassadors of a

resource R provided by x located at ring D as the nodes with ID y such that $y = \frac{R}{\Delta^{i-a}}$ where, $D-1 \leq a \leq D_{max}$. Let us illustrate this by means of the example of a resource with GUIDE $R = 112101001221$ provided by the node $x = 2$. The nodes of the set $\{1, 11, 112, 1121, ...\}$ are the ambassadors of x for the resource $R = 112101001221$ as shown in Figure 4.6. We can resume by stipulating the following lemma:

Lemma 4.5.1 *For any resource $R = r_{i-1}...r_0$ provided by the node x, there are at least* 1 *and at most D_{max} ambassadors for that resource in the network. With $2 \leq \Delta \leq N$ and the total number of rings forming the network is D_{max}.*

Proof: The number $\frac{R}{\Delta^{i-a}}$ is the identifier of exactly one node on the ring where x is located: when $a = D$. As the network grows, the outer rings are filled and there exists exactly one node with ID $A = \frac{R}{\Delta^{i-D-j}}$, with $0 < j < D_{max} - D$, on each ring between D and D_{max} such that A is ambassador of R. Thus, in a network with a total number of D_{max} rings, each node x located on the ring with ID D has an ambassador for a resource R on all the $D_{max} - D + 1$ rings with identifier larger than D.

Hence, any node x has at least one and at most D_{max} ambassadors in the network. ■

The resource distribution algorithm is depicted in pseudo-codes in Listing 4.1.

4.6 Routing in CMR

The routing operation in CMR is realized using three routines as described below. While routing a message toward a node on the same ring, the primitive *on_ring_forwarding* is applied. To route a message toward a node on an outer ring, the primitive *outwards_forwarding* is used. To route a message toward a node on an inner ring, the primitive *inwards_forwarding* is applied.

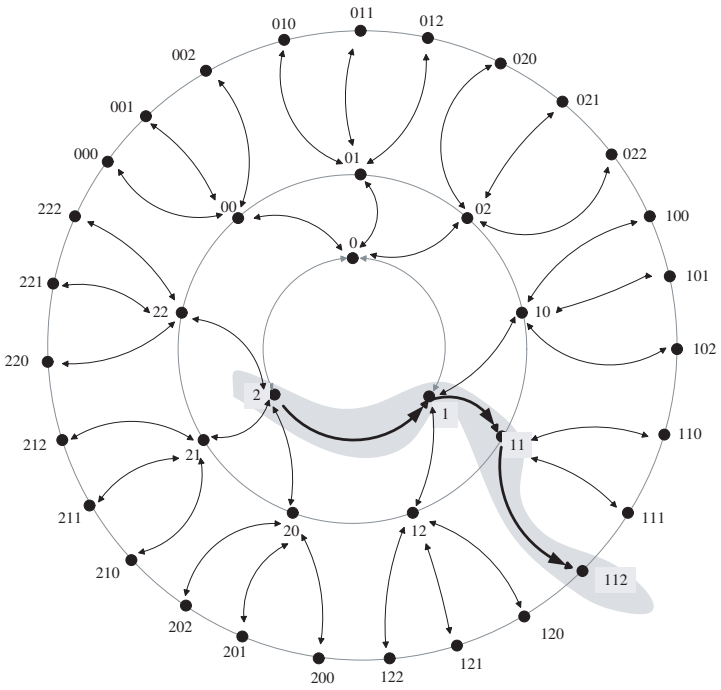

Figure 4.6: The resource with a GUIDE equal to 112101001221 located at the node with ID 2 has $1, 11, 112, ...$ as ambassadors. The ambassadors have the same prefix as the GUIDE of the resource.

Listing 4.1: Resource distribution algorithm: a node x publishes its resource with GUIDE R in the network. The procedure *handle*() is used for handling a resource after it is received from another peer.

```
Procedure x.distributeResource(R)
  if (x == ''new node in the network'' or x == ''new node in the
      ring'') then
    send(R, $r_{i-1}r_{i-2}...r_{i-D+1}$)
  return 0

Procedure x.handle(R)
  if ($r_{i-1}r_{i-2}...r_{i-D}$ exists) then
    send(GUIDE, $r_{i-1}r_{i-2}...r_{i-D}$)
  return 0
```

4.6.1 The On_ring_forwarding Routine

The *on_ring_forwarding* operation from a node x toward a node y uses the De Bruijn routing technique as described in Section 4.2. A node $x = x_{i-1}x_{i-2}...x_1x_0$ routes a message toward $y = y_{i-1}y_{i-2}...y_1y_0$ by forwarding to the node $z = x_{i-2}...x_1x_0y_{i-1}$ on the same ring. Hence, according to the De Bruijn properties, the longest path between any two nodes x and y on the same ring using the *on_ring_forwarding* primitive is of size i. The parameter i defines the diameter of the ring, where it is located.

4.6.2 The Outwards_forwarding Routine

If a node $x = x_{i-1}x_{i-2}...x_1x_0$ located on the ring with identifier $i = D_x$ is routing a message to a node y located on the ring with identifier D_y such that $D_x < D_y$, then the *outwards_forwarding* is applied. If the expressions $x_{i-1}x_{i-2}...x_1x_0$ and $y_{j-1}y_{j-2}...y_{j-i}$ are similar then the message is forwarded to the node $y_{j-1}y_{j-2}...y_{j-i}y_{j-i+1}$ on the next outer ring, else the *on_ring_forwarding* is used to forward the message to the next hop on the same ring. When $x_{i-1}x_{i-2}...x_1x_0$ and $y_{j-1}y_{j-2}...y_{j-i}$ are similar, the forwarding operation toward the destination is completed within at most $D_{max} - 1$ steps; with $D_{max} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ and N the maximal estimated number of nodes in the network.

Let us consider an example of two nodes $x = 110$ and $y = 11020112$. A message to be routed from x to y is first forwarded to a node $z = 1102$ on the next outer ring and then to the node $z' = 11020$, etc. until the node y is reached.

4.6.3 The Inwards_forwarding Routine

If a node $x = x_{i-1}x_{i-2}...x_1x_0$ located on the ring with identifier $i = D_x$ is routing a message to a node y of length D_y such that $D_x > D_y$, then the *inwards_forwarding*

is applied. The node x forwards the message to a node z with $z = x_{i-1}x_{i-2}...x_1$. The total number of hops in a CMR overlay with D_{max} number of rings is $D_{max}-1$ with $D_{max} = \log_{\Delta}(N(\Delta-1)+\Delta)-1$ and N the maximal estimated number of nodes in the network.

Let us illustrate this once again with the example of two nodes $x = 11020112$ and $y = 110$. A message to be routed from x to y is first forwarded to a node $z = 1102011$ on the next inner ring and then to the node $z' = 110201$, etc. until the node y is reached.

4.6.4 A Comprehensive Routing Operation

We propose a comprehensive routing operation between any two nodes $x = x_{i-1}x_{i-2}...x_1x_0$ and $y = y_{i-1}y_{i-2}...y_1y_0$, where x is the source and y is the destination. The routing operation from node x to node y is processed in the following steps:

4.6.4.1 For $D_x \leq D_y$

The source node x resides on an inner ring of the ring where the destination node y is located. The routing from x to y is performed in two steps: (i) *on_ring_forwarding*, followed by (ii) *outwards_forwarding* when the $D_x < D_y$.

On_ring_forwarding: x routes the message to the De Bruijn neighbors on the same ring by left shifting. The next-hop node $x_{D_x-2}...x_1x_0y_{D_y-1}$ also forwards until the message reaches the node $\bar{y} = y_{D_y-1}y_{D_y-2}...y_{D_y-D_x}$. Forwarding from x to y using *on_ring_forwarding* routine requires at most D_x hops.

Outwards_forwarding: after reaching $\bar{y}$, the message is forwarded outwards to the node with ID $y_{(D_y-1)}y_{(D_y-2)}...y_{(D_x-1)}y_{(D_y-D_x-i)}$ on the next outer ring by

incrementing i, — initially set to 1, — by 1 after each outwards routing step, until $i = D_y - D_x$ and the node $y = y_{(D_y-1)}y_{(D_y-2)}...y_1y_0$ is reached. The outwards routing is achieved in $D_y - D_x$ overlay hops.

In summary, the total number of steps necessary to move from any node x on an inner ring to any node y on an outer ring is at least $D_y - D_x$ and at most D_y routing steps.

4.6.4.2 For $D_x > D_y$

The source node x is located on an outer ring from the destination node y. Here, the routing is two steps: (i) *inwards_forwarding* followed by (ii) *on_ring_forwarding*.

Inwards_forwarding: the message is consecutively forwarded from the outer to the inner rings to the nodes $x_{D_x-1}x_{D_x-2}...x_i$ by incrementing i, — initially set to 1,— by 1 after each step, until the node $\bar{y} = x_{D_x-1}x_{D_x-2}...x_{D_y-1}$ is reached. The message arrives at the node $\bar{y}$ after $D_x - D_y$ steps.

On_ring_forwarding: when the message reaches the node $\bar{y}$ on the same ring as y, *on_ring_forwarding* is used and the destination y is attained within at most D_y steps.

The total number of steps necessary to move from any node x on the outer ring to any node y on the inner ring is at least $D_x - D_y$ steps and at most D_x steps.

A special case, however, is when a message is routed between two nodes x and y on the outermost ring of the network. The node x forwards the message through *inwards_forwarding* in one hop to node $\bar{x} = x_{D-1}...x_2x_1$ on the next inner ring. The node $\bar{x}$ routes the message toward the node $y^{in} = y_{D-1}...y_2y_1$ on the same ring. And finally, y^{in} forwards the message outwards in one hop to y on

the outer ring. Therefore, the routing operation from one node on the outermost, eventually incomplete, ring to another node on the same ring can be done along a routing path of length $D_{CMR} + 1 = \log_{\Delta}(N(\Delta - 1) + \Delta)$.

Lemma 4.6.1 *Routing between any two nodes in CMR can be achieved in at most $D_{CMR} + 1$ hops.*

Proof: Proof of Lemma 4.6.1 follows from the routing algorithm described above. ■

The complete routing algorithm is depicted in pseudo-codes of Listing 4.2.

4.7 The Resource Lookup Procedure

The principle is to find the provider or the replicator of the desired resource in the network. When a node y located on the ring with ID D needs a resource R, it sends a request message to the node $z_1 = \frac{R}{\Delta^{i-D+1}}$. If y is located on the innermost ring, it sends a request message to the node $z_1 = \frac{R}{\Delta^{i-D}}$ on the same ring. If z_1 is not aware of x's location, it forwards the query message to its next outer ring. A node $z = \frac{R}{\Delta^{i-a}}$ with $D - 1 \leq a \leq D_{max}$ is likely to know the location of x: the provider of R. According to the resource distribution procedure, there is at least one node z that is an ambassador of the resource R. The chance to find a node that is aware of x's location grows as the query is forwarded to the outer ring. This is because a resource $r_{i-1}r_{i-2}...r_1r_0$ is always published to at least one node closest to the outermost ring.

As a direct consequence of the resource distribution presented in Subsection 4.5 and the resource discovery described here, we stipulate the following lemma:

Listing 4.2: Routing algorithm: route from a node x to a node y on the CMR overlay.

```
/***
 * Inputs: x = source; y = destination
 ***/
Procedure route(x,y)
  /* function ring(x) returns the actual ring number for node x.
     ring(0)=ring(1)=1 */
  if ring(x) == ring(y) then
    /* Check whether the nodes are on the outermost ring */
    if ring(x) == D_max then
      return inwards_forwarding(x, y)
    else
       /* The nodes are on a lower fully constructed ring */
      return on_ring_forwarding(x,y)

  else if ring(x) < ring(y) then
    if prefix(y) == x then
      return outwards_forwarding(x,y)
    else
      return on_ring_forwarding(x, y)

  else if ring(x) > ring (y) then
    return inwards_forwarding(x, y)

/***
 * The procedures return the next-hop on the way from x to y
 **/
Procedure outwards_forwarding(x, y)
  int D = ring(x) + 1;  // the ID of the x's next outer ring
  return (y modulo (\Deltaˆ D));

Procedure inwards_forwarding(x, y)
  int D = ring(x) - 1; // the ID of the x's inner ring
  return (y modulo (\Deltaˆ D));

Procedure on_ring_forwarding(x, y, i)
  int \Delta; // The constant degree of the De Bruijn graph for the
     x's ring
  int N'; // The number of nodes on the ring
  int k = top(y, i); // The i-th letter of y counted from left
  i++;
  return (dx + k) modulo N';
```

Lemma 4.7.1 *The resource lookup operation for a desired resource R in the network can find an ambassador in the network within at most*

$$D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$$

overlay hops.

Proof: The Lemma 4.6.1 states that the route path-length from any node x to any node y is at most $D_{CMR} + 1$. However, if a resource R exists in the network, its furthest provider can be located after at most $D_{CMR} + 1$ forwarding hops. Moreover, according to the Lemma 4.5.1, if its resource provider is located on the outermost ring with identifier $D_{CMR} + 1$, it may have chosen an ambassador on the next inner ring.

In conclusion, a node $y = \frac{R}{\Delta^{i-a}}$, where $D - 1 \leq a \leq D_{max}$, is reached after at most D_{CMR} hops, which has a location table entry with the resource R. ■

4.8 Summary and Conclusion

This chapter answers the question on how resources in the network are organized. It provides a strategy that efficiently managed the nodes and resources of the network such that they can always be found with high probability within the shortest path even when there are failures and unpredictable joins. We have proposed a combination of a concentric multi-ring overlay structure and an adaptation of De Bruijn digraph concept to achieve a highly interconnected P2P system. The proposed architecture presents three major advantages for resource management in dynamic distributed systems:

- ◇ the flexible object distribution which is only dependent on the global unique resource identifier (GUIDE). It enables any node in the network to find

the desired resource with high probability, when the resource exists in the network. This property enables the lookup of resources in at most $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ overlay hops, which is the lowest bound compared to other outstanding P2P proposals.

- ◇ the fixed degree $2\Delta + 2$ of the nodes in the network, which is independent from the total number N of nodes. This property has the advantages that the increase of the network size does not necessarily increase the size of the control overhead generated to keep track of the neighborhood which is the case in related approaches such as Chord [113], Pastry [99] and Tapestry [126].

- ◇ The range query can be supported, since there are more than one node responsible for a given resource in the network. This property is important for guaranteeing fault-tolerance as we shall see in the next chapter.

The question on how the system behaves in situations of churns will be studied in more detail in the next chapter.

Chapter 5

A Churn-Resistant Strategy for the CMR Overlay

IN the previous chapter, we have proposed a network organization strategy that represents the nodes of a P2P system on a concentric multi-ring overlay. The nodes on the overlay choose their neighbors based on the De Bruijn digraph concept and each node can route to at most $2\Delta + 2$ neighbors within at most D_{CMR} overlay hops. The resource distribution strategy is based on a slightly modified version of the DHT concept, where each data item is assigned more than one ambassador. To work properly, the overlay is so constructed that it is always in a consistent state. In fact, the overlay is in a consistent state when the following conditions are fulfilled:

1. **Fully constructed rings:** each inner ring with ID D is always fully constructed with Δ^D nodes.

2. **Ring contact node:** each ring has a ring contact's node $x_{\{init,D\}}$.

3. **Head node:** each node in the network has one head node.

4. **Consistent routing table:** each node in the network has at least one and at most $2\Delta + 2$ neighbors.

5. **Ambassadors:** each resource is replicated to at least one and at most D_{max} ambassadors in the network.

6. **Uniqueness of ID:** any two different nodes on the overlay have different IDs.

However, since a node can leave or join unpredictably at any time in a P2P system, ensuring an always consistent state of the overlay is not a trivial task. Because of the network's churns, — the events of node joining and leaving in the network,— an inner ring will either become incomplete or a ring contact node will be deleted unpredictably, or a node will join the network with some new resources. To keep the network in a consistent state, a churn-resistant strategy is required to detect new resources and node's departures and to repair the resulted inconsistency.

This chapter describes a churn-resistant strategy for keeping the CMR overlay in a consistent state each time new nodes join or leave the network. The strategy consists of three parts: (i) the node's insertion algorithm to manage new node or new resources in the overlay structure, (ii) the node deletion algorithm to recover from failure, and (iii) the static resilience algorithm for alternative routing during node failure.

The strategy aims at enabling network consistence using a low-cost maintenance concept while preserving load balancing. The object lookup operation should be achieved within the shortest possible path even in situations of high failure rate or concurrent joins in the network.

5.1 Isolated Node's Insertion

The concentric multi-ring evolution assumes that new nodes can join the system at any time, hence increasing the number of participants in the system. This has two major impacts on the network management operation. The first one is positive, because the greater the number of nodes in the network is, the larger is the number of resources and the number of routes in the network. The second impact is problematic because the overall network management cost increase with each new node in the system. In fact, the neighborhood information messages must be exchanged between the system and the new node before its new resources are published. With each new node in the network, at least one new resource is available on the overlay. This new resource should be published to at most one ambassador to avoid network's inconsistency. In the following, we discuss in more detail the three phases for the insertion of a new node in the network.

5.1.1 The Bootstrapping Phase

In the bootstrapping phase, a node initiates the first contact with the system. A node must know the IP address of at least one other node already in the network, the so-called network contact node to bootstrap the registration in the network. But, how does the new node know a contact node in the system? In most of the cases, this is usually done by providing a fixed, well-known server, also called bootstrapping server [1, 4, 8], that serves a list of peers already in the system. However, this concept is against the principle of P2P, where the whole system is assumed to be self-organized without any central instance. Another possibility is to use connections to random IP addresses in a certain range of addresses where there is a good chance of getting a connection.

Therefore, since we are not directly focusing on the bootstrapping problem

in self-organized systems in the thesis, we simply assume that a node x willing to join the network is aware of at most one node already present in the network, which it uses as network contact node. Throughout this document, we assume that the network contact node is the node with ID $x_{\{init,1\}} = 0$, the first node on the innermost ring. Thus, the churn-resistant strategy presented here is not dependent on any bootstrapping strategy.

According to Construction rule 3 (on page 78), a newcomer node joining the network is always represented on the outermost ring of the overlay, because any inner ring is always fully constructed. Thus, inserting a new node in the network can be reduced to: (i) determining the ring identifier of the outermost ring, and (ii) assigning a node's identifier to the newcomer on the outermost ring.

5.1.2 Determining the ID D_{max} of the Outermost Ring

A node must send a *join_request* message to the network contact node to ask for the attribution of an ID on the outermost ring. The *join_request* message is the first message a node sends when it is willing to join the network. This message is routed using *outwards_forwarding* routine from the node $x_{\{init,1\}} = 0$ to the node $x_{\{init,D_{max}\}}$ on the outermost ring. The *join_request* message reaches the outermost ring contact node in no more than D_{max} hops involving at most D_{max} nodes in the joining process. Once being on the outermost ring, the message is handled only by a subset of nodes that decide, which identifier is to be attributed to the new node.

5.1.3 Assigning an ID to a Node Joining the Network

When the *join_request* message reaches the ring contact node $x_{\{init,D_{max}\}}$ on the outermost ring, the node $x_{\{init,D_{max}\}}$ can either decide to assign an ID to x or to

forward the message to other nodes on the outermost ring. The procedure of ID assignment must ensure that: (i) there is no duplication of IDs on the ring, and (ii) only a small subset of the nodes in the network are involved in the process. This is important for scalability reasons since many nodes may join the network regularly.

For efficiency reasons, we have introduced the notion of branchtable in the ID's assignment procedure. By definition, each node on a ring maintains a branchtable, which consists of at most Δ branches, Δ being the number of De Bruijn neighbors of the node. One branch is maintained for each De Bruijn neighbor. A branch can be considered as a data structure containing the identifier x^{DB} of the De Bruijn neighbor and an integer $i = \frac{\Delta^{D-a}-1}{\Delta-1}$, where a is the position of the last occurrence from right of a non zero digits sequence in the Δ-base expression of x, counted starting from 0. If x is zero then the position $a = 0$. The value i is also called the branchsize. For example, the position of the integer 012 of base 3 is $a = 2$.

The branchtable $BT(x)$ of a node x is formally defined as the set of tuples (x^{DB}, i) so that $x^{DB} \in \Gamma^+(x)$.

$$BT(x) \stackrel{déf}{=} \left\{ (x^{DB}, i),\ x^{DB} \in \Gamma^+(x) \mathit{and}\ i = \frac{\Delta^{D-a}-1}{\Delta-1}, \mathit{where}\ a \leq D \right\}$$

To better illustrate this, let us consider the example of the branchtable of the node $x = 001$ which is formed of the three tuples

$$BT(001) = \{(010, 4), (011, 4), (012, 4)\}.$$

Each time a node on the outermost ring receives a *join_request* message with no final destination, it forwards the message toward its De Bruijn neighbor that

has the branch with the largest integer value i and consequently decrements the integer value i of this branch by one. When the De Bruijn neighbor x^{DB} of a node x corresponds to the branch with the largest integer value i and the neighbor is not yet present in the network, the newcomer's identifier is x^{DB}. In other words, when the branch with the largest branchsize is equal to $i = \frac{\Delta^{D-a}-1}{\Delta-1}$ according to (5.1.3), then the new node is the node with identifier x^{DB}. The node x is called the head node of the node x^{DB} and it is per definition the node that sends the *join_invite* message to the newcomer with its new identifier. In this manner, when the branchsize of all the De Bruijn neighbors on a ring is set to zero, then the ring is said to be fully constructed and a new ring is created.

Lemma 5.1.1 *At any given time in a network without failure, the difference between the integer values i of any given two branches of a node x is at most* 1.

Proof: Lemma 5.1.1 follows from the explanation above. We have seen that the *join_request* message is always forwarded in the direction of the node with the largest branchsize followed by a decrement of the branchsize. ■

Once joining the network, the new node x^{DB} periodically sends one KEEPALIVE message to its head node x, which is used to ensure that x^{DB} is still present in the network. The head node which plays an important role in the ring's management strategy is sometimes denoted as x^{head} in this document.

In summary, the forwarding of a *join_request* message among the nodes on the outermost ring is achieved in at most D_{max} hops. Hence, each node joining the network involves at most $2D_{max}$ nodes to forward the join request message until the node is assigned an ID. After joining the system, the new node notifies its presence to its neighbors, distributes its resources, and collects notification messages received from other nodes to construct its tables. After that, the node is connected and can request and exchange resources with the rest of the network.

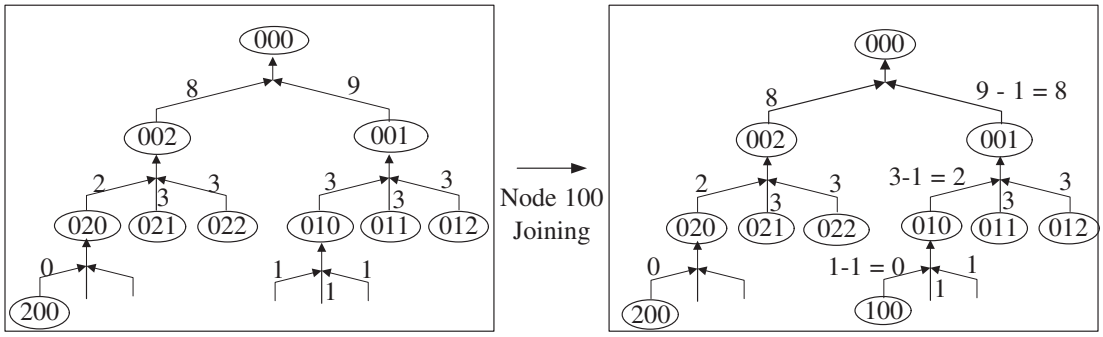

Figure 5.1: Insertion of the node 122

Each node joining the network generates one message to determine the outermost ring's identifier, one resource location message to distribute each of its resources, — according that each node provides one resource to the network,— and at most $\Delta + 1$ messages to update its routing table. This makes a total number of $\Delta + 3$ messages, which involves at most $2D_{max}$ nodes for forwarding.

5.1.4 The Join Procedure by Example

We consider the case of a node x joining the network on the outermost ring with identifier $D_{max} = 3$. The outermost ring of a concentric multi-ring overlay with $\Delta = 3$ is represented in the form of a tree architecture as shown in Figure 5.1. The root of the tree is the contact node of the ring with the identifier 000. We consider a network, where 10 nodes have already been inserted on the outermost ring and where 17 other nodes can additionally join such that the ring is fully constructed.

Before joining the overlay, the new node x is identified e.g. in the Internet using its IP Address. x sends a *join_request* message to the node with identifier 0 on the innermost ring of the overlay. It has been assumed earlier in this chapter that a node x knows the IP address of the node 0 on the overlay. On receipt of

the *join_request* message of x, the node 0 routes the *join_request* message to the node 000 on the outermost ring through 00. Recall that 0 has three neighbors on the next outer ring, the nodes 00, 01 and 02. When the join request message arrives to the node 000 on the outermost ring, it checks its branchtable to find out where to forward the message. The first branch contains information about the node with ID 001 and has a branchsize $i = 9$ and the second branch has information about the node with ID 002 with a branchsize $i = 8$. Thus, the node 000 decides per definition to forward the message to the node 001. The node 000 then decrements the branchsize of its branch with ID 001 by 1 to $i = 8$.

When the node 001 receives the message, it proceeds like the node 000 has done before and forwards the message to the node 010 with branchsize $i = 3$ and decrements i by 1 to $i = 2$. When the node 010 receives the *join_request* message, it proceeds on the same manner like the other two nodes 000 and 001. However, the difference is that the size of the branch with ID 100 is set to $i = 1$ which is equal to $\frac{\Delta^{D-a}-1}{\Delta-1}$. Hence, the node 010 sends a *join_invite* message to the node x, which can join the network overlay with the ID 100.

In conclusion, a node joining in the concentric multi-ring overlay is a scalable operation with a local impact and a constant management cost comparing to other outstanding P2P solutions such as Chord, Pastry and Tapestry that require around $\log N$ control messages, when a node is inserted the system. The performance of the joining operation is decoupled from the bootstrapping operation. Therefore, our assumption on the existence of a network contact node is valid and does not hide any complication for the management of the concentric multi-ring overlay using a specific bootstrapping technique.

5.2 Concurrent Join

When many nodes simultaneously wish to join the network at a given time, a concurrent JOIN management protocol is required. This is an interesting aspect in the management of dynamic distributed systems. If there are several access points in the overlay, the operation of concurrent join can become a critical one, because two nodes that join the network exactly at the same time t could be assigned the same ID. This is undesirable because the node's identifier is used on the overlay for the routing, the information distribution and the lookup of data. In case of an identifier's duplication, routing in the shortest path can not longer be taken for granted. This confusion on the nodes' ID may lead to uncertainty on the success of routing operations.

On the concentric multi-ring overlay, we avoid the identifier duplication as follows. Due to the fact that new nodes in the network join the overlay on the outermost ring and the network contact node is on the innermost ring, the concurrent joins might not cause any additional control overhead in the network. The problem of concurrent join is solved by serving concurrent *join_request* messages sequentially. That is, when several nodes want to join the network at the same time, they are served one at a time just like in a queuing process with a single server. Hence, when a subset of k nodes try to join the network from different locations at the same time, each node sends a *join_request* message to the network contact node. Once receiving the *join_request* message, the network contact node initiates the node insertion process as described in the isolated join procedure in Subsection 5.1. The subsequent processing of simultaneous *join_request* messages makes it impossible that two nodes are assigned the same identifier in the network.

However, a side effect of the subsequent processing of concurrent join requests

is the delay that can be observed by a node during the network admission phase. In fact, in a concurrent join procedure with sequential processing of join request messages, a node in the tail of the queue will wait longer than a node that is in the head of the queue (a kind of FIFO service).

5.3 Management of the Node's Failure

In dynamic distributed systems such as P2P networks, nodes are not centrally coordinated. They generally join the system to achieve an individual and specific goal whereby ensuring a global common management target. Moreover, the nodes of a P2P system are transient in the sense that they mostly participate in the system's operations only as long as they are looking for some resources, — e.g. in the case of file-sharing, a node joins the community to lookup for specific music files. The node is no longer part of the network when the file-sharing program is exited. Hence, a node or a set of nodes can fail or leave without prior warning. This transient and unpredictable behavior is problematic for the resource awareness in structured P2P systems, because the predefined structure will become inconsistent when some nodes leave. It is therefore necessary to develop strategies to keep the system in a consistent state in churn situations.

In this section, we differ between two cases of node's failure in the system. Firstly, when a node leaves the system after notification of its neighbors; in this case the notification message is interpreted as a BYE message. Once receiving the BYE message a recovery procedure is started. Secondly, when a node ungracefully or accidentally leaves the system; generally, it does not have time to send any BYE message to its neighbors. An efficient failure detection algorithm should be designed such that the failures are detected immediately and with the

lowest possible cost. A fast detection is advantageous because, the longer a network is in an inconsistent state, the higher is the probability that a message is routed through the failed non-existent node and the destination is never reached. Low-cost detection is also desirable, because in a failure-prone network the links between nodes in the network will become congested due to the many control messages.

In both failure situations, however, at least one path in the network will become invalid and the system will be in an inconsistent state for a while until the failures are recovered. The failure recovery strategy can efficiently help only if its complete procedure is as fast as possible.

5.3.1 Failure Detection Algorithm

In this section, we describe in more detail the procedure of failure detection in both cases: after sending a BYE notification and without sending a BYE notification.

5.3.1.1 Case 1: Leave after Sending a BYE Notification

The standard case of failure on the CMR is when a node x sends a BYE notification message to its incoming neighbors before leaving the network. The incoming neighbors of a node x are the nodes that have an entry for x in their routing table:

1. the head node of x: x^{head},
2. the incoming De Bruijn neighbors of x,
3. the outer ring neighbors of x, if they already exist, and
4. the inner ring neighbor of x.

Since the IP address of the incoming De Bruijn neighbors are not comprised in the table of x, x uses their identifiers to contact them. After reception of the BYE message from x, each node deletes the entries for x in their location and routing tables. Thus, a message destined in the direction of x can be routed along an alternative path, until a new node x is designated. The operation of the alternative routing is handled later on in this chapter in Section 5.4. The head node, which receives this message can react by choosing, — when necessary,— a successor for the leaving node. As an optimization, a node that leaves the network can also send the state of its routing and location tables to its head node, which uses them to initialize the tables of the successor. The advantage of sending a BYE notification is that the failure detection is achieved almost instantly.

5.3.1.2 Case 2: Leave Without prior Warning

On the other hand, the failure detection, when a node ungracefully leaves the network, is a little bit more complex. In fact, each node sends one periodic heartbeat message (also called KEEPALIVE) to its head node on the ring[1]. When a head node does not receive any KEEPALIVE message from a given node after a predefined timeout interval T_{out}, it can assume that the node has probably crashed and starts the network recovery procedure. We say "probably", because it could happen that x is still available in the network but the link between x and x^{head} is congested or cut for some physical reasons. To detect such cases of link failure, a false-positives detection procedure can be used. The detection of false-positives is not handled here because it exceeds the scope of this thesis. After the failure detection is confirmed, the head node x^{head} starts a recovery procedure.

[1]Other P2P approaches let the nodes in the system trace the entire routing table in a periodic fashion.

The timeout value T_{out} and the frequency T of sending KEEPALIVE messages are significant parameters that can be modified to tune the performance of the failure detection algorithm. The choice of T and T_{out} is conditioned by the dynamic pattern of the network. For example the measurement studies presented in [54] show that the average session lifetime of the nodes in a KaZaA P2P system is around $28,25$ minutes for the 99^{th} percentile of the nodes in the network. One can choose a value for T depending on the type of application and the trade-off between the number of tolerated failures and the available bandwidth for control messages. The study presented in [96] demonstrates that the choice of the timeout value T is a significant decision for the lookup latency under churns.

5.3.2 Recovery from a Voluntary Node's Departure

After a failure has been detected, the recovery procedure must ensure that the network remains in a consistent state. In the CMR overlay a clean-up operation is executed to update the tables of the nodes that have x in their routing tables as described in Subsection 5.3.1. When necessary, the head node starts a *choose_successor* operation to select a successor for the leaving node.

5.3.3 Recovery from an Ungracefully Node's Failure

Against the case discussed above, the nodes in dynamic distributed systems can also leave the network without any prior warning. This is a particularly challenging issue, because one can not predict such an event and the time until the failure is detected and recovered is critical for the system efficiency. Therefore, for the management of nodes' failures on the CMR overlay, we have developed a strategy to detect failures in an optimal way, and to recover from failures before messages get lost in the network.

5.3.3.1 Failure Recovery Algorithm

Once a head node x^{head} has got a failure confirmation for a node x, two operations are processed:

Firstly, the node x^{head} sends a *route_obsolete* message to all the nodes, which have x as a forwarding hop. Once receiving a *route_obsolete* message, the nodes update their routing and location tables. This operation is to avoid that nodes continue to forward message toward x, since it will be lost.

Secondly, the node x^{head} sends a *choose_successor* message to select a successor for the left node x. The *choose_successor* message is forwarded to the outermost ring's contact node, which routes it to its branch with the smallest size. The operation of choosing a successor is described below in Section 5.3.4.

Once on the new ring with a new ID, the node subsequently informs its ongoing De Bruijn neighbors on the same ring. It also informs its inner ring's neighbor and its Δ neighbors on the next outer ring about its new IP address. This stabilization process enables the network to converge very fast and generates only $2\Delta + 2$ messages. To complete the recovery procedure, the new node x chooses one ambassador for each of its resources.

5.3.4 Procedure for Choosing Successor

The procedure for choosing successor describes how to select a successor for a leaving node x on the overlay. This procedure depends on the last position of the left node on the overlay:

- ◇ if x was on an inner ring of the overlay, then it should be replaced because each inner ring must always be fully constructed to ensure network consistency.

- ◇ if x was on the outermost ring and the sum of the branchsize of x^{head} is $i < \Delta * \frac{\Delta^{D-a}-1}{\Delta-1}$, then it should also be replaced, else the position of x can be left vacant.

The successor is always chosen from the outermost ring. Here, we also use the concept of branchtable like in the join procedure in the previous Section 5.1. When the head node x^{head} of the node x receives a BYE message, it checks if the node should be replaced or not. If the node must be replaced, then the head node x^{head} that has detected the leaving intention sends a *choose_successor* message on the outermost ring to the root node $x_{\{init,D_{max}\}}$ of the tree representing the outermost ring. On receipt of the message, the node $x_{\{init,D_{max}\}}$ forwards it to its De Bruijn neighbor or tree's branch with the lowest branchsize and increments the branchsize in that direction by one. The De Bruijn neighbor receiving the message also forwards it to its neighbors with the lowest branchsize (incrementing the branchsize in the forwarding direction) until the message reaches a node that has no node in its branches. Then, this node is chosen as the successor of x and changes its identifier, and eventually its ring on the overlay.

The new node x differs from the old one in two points: (i) the IP address, and (ii) the provided resources. The new node with identifier x has the routing and the location tables of the old node x. For consistency reasons, once on the new ring the new node publishes its resources to the ambassadors on the next inner ring or to the ambassadors on the same ring. Moreover, it informs its Δ incoming De Bruijn neighbors, its Δ outer ring neighbors and its inner ring neighbor. All these neighbors update their routing tables with the new IP address of x. This operation implies a network maintenance cost of at most $2\Delta + 2$ messages for each node leaving the network.

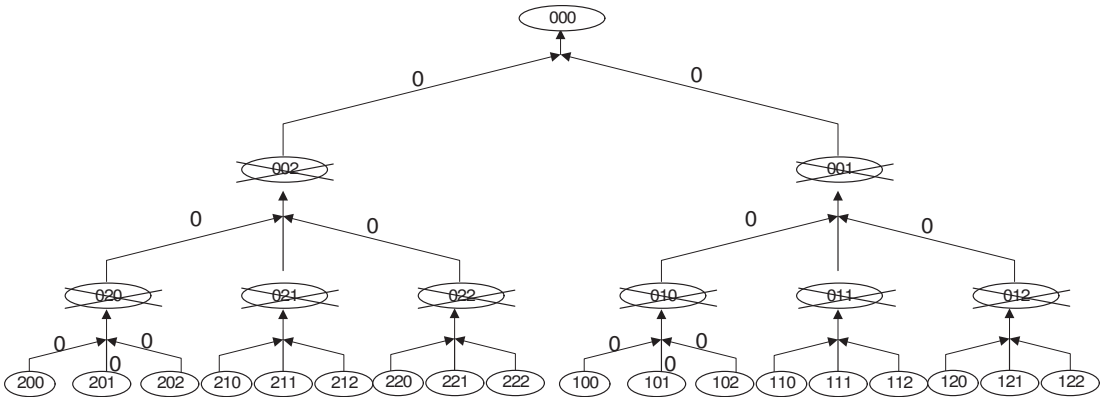

Figure 5.2: Case study for concurrent failures on the same ring

5.3.5 Concurrent Failures Event

The recovery procedure after a concurrent failure event is an exciting issue, which is mainly handled like the individual failure event described above. According to the dynamic network experiments [79, 88], where the node's departure event is modeled as a Poisson distribution, the probability of k collective nodes' failures decreases exponentially as k linearly increases. On the CMR overlay, several nodes could fail from the same or from different rings. The recovery from a collective failure event is an extension of the individual failure recovery procedure. The difference resides in the failure detection procedure. To better understand this issue, we discuss in the sequel three possible scenarios.

Case 1: Failure of all the nodes in the neighbor table If all the $2\Delta + 2$ neighbors of a node in the network failed simultaneously, then the node will definitely be disconnected even for a while until the network is reconfigured. When the entire routing table of a node is faulty[2], which is one of the worst case scenarios, the node can use the entries in its location table to bootstrap again in the network. When disconnected because all its outgoing neighbors have failed, the node arbitrary chooses one node from its location table that it uses as

[2]Although this is an almost unlikely event.

bootstrapping node to reconstruct its table and relocate itself in the overlay.

Case 2: What happens when several nodes on a ring concurrently fail? Let us consider the illustration of Figure 5.2, where 8 nodes on the ring with ring's ID equal to 3 have failed concurrently. The ring's contact node with ID equal to 000 will detect after a timeout t that the nodes with ID equal to 002 and 001 have left and it will start a recovery procedure to replace these nodes. After successfully replacing the nodes 002 and 001, the node with ID 002 will detect that it does not have any De Bruijn neighbor and that the branch size of the node with ID 000 in the direction of 002 is different of the normal value that should be 11. The new node with ID 002 will conclude that the branch size of the node 000 is abnormally set to 0, which means that at least the nodes with IDs 020, 021 and 022 have also left. Consequently, 002 will start a recovery procedure just as the node 000 has done before. The node 001 will also do the same and all the failed nodes on the ring will be replaced.

Case 3: What happens when all the nodes on a ring concurrently fail? Still keeping the illustration on Figure 5.2 in mind, let us have this time the situation where all the nodes have failed concurrently. The node's contact ring of the next inner ring, — the node with ID 00 which is the head node of the node with ID 000,— will detect that the node 000 has left and will start a recovery procedure. The node with ID 00 will send a message to its neighbors to detect the address of at least one node on the outermost ring. This is possible since some inner ring nodes were located on the outermost ring before migrating to an inner ring. Such a node could still holding some resource information for providers that are probably on the outermost ring. This is possible when we assume that after changing the ring, a node does not release the old resource information

in its resource location table. Using these information, the node with ID 00 is susceptible to find a node of the outermost ring that knows the location of the outermost ring's contact node. The latter will then choose a successor for the node with ID 000. Once replaced, the node with ID 000 will also choose a successor for the nodes 002, 001, and so on.

5.3.6 Results

The node deletion management above ensures that any inner ring is always fully constructed in the network. As a conclusion, we can stipulate the following:

Lemma 5.3.1 *Each ring with identifier D, so that $D < D_{max}$ is with high probability (w.h.p.) always fully constructed, with the ring capacity $N = \Delta^D$.*

Proof: According to the network maintenance rules, the only ring with the eventual incomplete ID space is the outermost one with the identifier equal to D_{max}.

When the stabilization routine is processed, an inner ring could be in an incomplete state. However, if the stabilization routine takes an extremely short time τ to complete its task in a network with a lifetime T, then an inner ring in the network is inconsistent with probability $\frac{\tau}{T} \approx 0$. This means that any inner ring is always fully constructed w.h.p. ■

5.4 Static Resilience

The routing between any two nodes x and y in the network is a multi-hop forwarding operation. However, the routing operation described in Section 4.6 assumes that the network is in a consistent state and will consequently fail to route packets to its destination when a neighbor z along the path from x to y suddenly fails. To

address this issue, the P2P application based on the CMR overlay will be notified by a failure message when the route is not available. The application must be designed such that it retries the forwarding operation after a failure notification, with the chance that the route entry exists in the routing table.

Theorem 5.4.1 *Any resource request in the network is forwarded along the shortest path to its destination, even at a high rate of node failure.*

Proof: The static resilience scheme in this subsection ensures that a packet can always be routed to its destination although with a small delay for retry. Since, each failure can be detected after a given time interval in the network, we can guarantee that any failure will always be recovered and that the selection of next-hop for message forwarding is always guaranteed, as specified in Section 4.6. Additionally, according to the Lemmata 5.3.1 and 4.7.1, we can conclude that any request in the network is always forwarded along the shortest path to its destination even in failure situations. ■

5.5 Summary

In this chapter, we have presented a management strategy to detect failures and to re-establish consistent state of the concentric multi-ring overlay. We have studied some leave and join scenarios: (i) for concurrent nodes joining, (ii) for concurrent nodes leaving, and (iii) for nodes joining and leaving simultaneously. We have shown that even in the extreme situation, when more than half of the nodes in the network leave concurrently, the network converges into a consistent state after a finite time and messages can be routed to the destination within at most $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ hops.

Chapter 6

A Theoretical Evaluation of HiPeer

WE have presented in the Chapter 4 a concentric multi-ring architecture that defines how the nodes are interconnected and how resources are published and discovered in the network. As a management extension, we proposed in Chapter 5 a strategy for maintaining the network in a consistent state even during high rate of churns. The combination of both the CMR overlay and the churn-resistant strategy forms what we name **HiPeer**, a highly reliable P2P system.

This chapter provides answers to the following two questions about the performance of HiPeer:

1. how good does a CMR-based P2P system perform with regard to the management cost, the lookup path length, the load distribution and the lookup success percentage.

2. is the proposed structured architecture suitable for resource management in P2P systems.

Here, we analyze the impact of the dynamic behavior of the network on the performance of HiPeer. On behalf of a network dynamic model, we evaluate the performance of the proposed approach according to requirements of Section 1.2 in Chapter 1.

6.1 The Analytical Model

We consider a P2P network, where nodes can join and leave individually and collectively. We model the node's join and leave activities as a succession of Poisson distributed events. Hence, the probability P that a subset of k nodes in a network collectively join or leave the network within a given time interval is equal to

$$P\{x = k\} = e^{-\lambda}\frac{\lambda^k}{k!} \quad k = 0, 1, 2, ..., \infty$$

where,

- e is the base of the natural logarithm ($e = 2.71828...$),
- $k!$ is the factorial of the integer number k,
- λ is a positive real number, equal to the expected number of occurrences of the event during that time interval.

For instance, if a join or leave event occurs on average every two minutes, and we are interested in the number of events occurring in a ten minutes interval, we would use as model a Poisson distribution with $\lambda = 5$. The Poisson distribution assigns probabilities to the number of occurrences of a certain event. It is commonly used to model the number of random occurrences of some real world phenomenon in a specified unit of space or time. In fact, the study of the nodes' activities in P2P systems can be well-modeled using Poisson. The results of the

measurement experiment presented in [106, 54] confirm that the event of nodes joining and leaving in a P2P system can be approximated by using the Poisson distribution model.

For a realistic analysis of our P2P approach, we assert that it is unlikely or almost impossible that more than half of the nodes in the network ($\frac{N}{2}$ nodes) could leave collectively. The collective leave event of half of the nodes in the network is also known as the half-life phenomenon. The half-life phenomenon has also been used to analyze the evolution of the Chord protocol in [79]. This assertion is derived from the Chebyshev Inequality theorem [89] which stipulates that the probability of occurrence of an event x, where x is outside an arbitrary interval $(\eta-\epsilon, \eta+\epsilon)$, is negligible if the ratio $\frac{\sigma}{\epsilon}$ is sufficiently small. The expression σ^2 is the variance of a Poisson distributed random variable x. The Chebyshev Inequality is formally defined as follows:

Definition 1 *Chebyshev Inequality*

$$\textit{For any } \epsilon > 0,$$
$$P\left\{|x-\eta| \geq \epsilon\right\} \leq \frac{\sigma^2}{\epsilon^2}$$

In fact, if $\epsilon = \frac{N}{2}$ and $\sigma^2 = \eta = \lambda$, then the probability that more than half of the nodes in the network collectively leave, is

$$P\left\{|x-\eta| \geq \frac{N}{2}\right\} \leq \frac{4\lambda}{N^2}.$$

However, with a large value of N, the expression $\frac{\lambda}{N^2}$ is sufficiently small and tends to zero. Therefore, the probability that more than half of the nodes in the network collectively fail, is almost zero. The probability of the half-life phenomenon constitutes an important foundation for the following discussions in this chapter.

6.2 Network Consistency

The CMR network should be maintained in a consistent state in order to ensure good lookup performance, load balancing and highly availability. Using the half-life phenomenon presented above, we argue that even when half of the nodes in the network fail, the network always converges into a consistent state with high probability. In fact, when $\frac{N}{2}$ nodes leave collectively, approximately half of the entries in the table of each node remaining in the system would be out-to-date. Because the neighbors on the ring are geographically widely distributed, we can assume that even in the worst case at least $\frac{1}{3}$ of the table of each node remaining in the network will still be valid after the half-life phenomenon. That is, each node on the CMR overlay would have at least $(2\Delta + 2)/3$ valid table entries even after half of the nodes in the network have failed.

Lemma 6.2.1 *The CMR network is always in a consistent state with high probability, even when half of the nodes in the network fail simultaneously.*

Proof: This lemma mainly follows from the two Construction rules 1 and 2 (on Page 77) and the assertion that at least $(2\Delta + 2)/3$ neighbors are still alive even after $\frac{N}{2}$ collective failures. If at least $(2\Delta + 2)/3$ neighbors of each node are still alive after $\frac{N}{2}$ failures, then the head node of at least one failed node will be alive to detect the failure of at least one failed node. After detecting the failure, the head node starts a stabilization routine, which let the network converge into a consistent state after a finite time. In the case of concurrent failures, a stabilization routine will engender other stabilization routines until all the incomplete inner rings are fully reconstructed and each node of the outermost ring is connected to a head node.

Besides, given a parameter τ as the time needed to recover from a failure and a parameter T defined as the system lifetime since no collective failures have been

registered, we can assert that the probability that a lookup message is not routed correctly to its destination is $P = \tau/T$. However, since an individual failure is generally directly recovered and the probability of the occurrence of collective failures is lower than the probability of the occurrence of an individual failure, it can be deduced that the value of T is too large comparing to τ and $P \approx 0$. $P \approx 0$ implies that the network is with high probability always fully constructed and consistent. ■

According to the ability of the churn-resistant strategy to support fast network reconstruction in failure situations, routing of messages between any two points x and y on the CMR overlay can always be successful with high probability. That is, a message routing operation can fail only when a message is sent to a failed node at a time within the interval $[t, t + \tau]$, τ being the time duration for the network recovery. Even in such a situation, the message will not be lost because, the P2P application will retry after a given timeout $T > \tau$ with the certainty that the failure has already been recovered. From this argumentation and according to the requirement for fault-tolerance in Chapter 1, we can deduce that:

Theorem 6.2.1 *The CMR overlay is fault-tolerant in the sense that, even when many failures occur collectively, the message routing operation between any two nodes in the system can always be successfully routed toward its destination.*

Moreover, when nodes join and leave continually the system, the network can become disconnected if a threshold number of k nodes is deleted. This threshold is known as the connectivity degree of the network, which is a key factor for determining the reliability of the nodes' interconnections. The connectivity is defined as the cardinality of the minimal subset of nodes disconnecting the network, when it is deleted [78]. Studying the performance of CMR, we can state the following:

Lemma 6.2.2 *If the number of nodes in the network is at least Δ, each node can always find a path to any other node in the network.*

Proof: The proof that the network is connected at any given time with high probability is based on three properties of the overlay networking topology: (i) it has been proved in Lemma 5.3.1 (on Page 110) that any ring with identifier $D_{max} - 1$ is always fully connected, (ii) Lemma 4.5.1 (on Page 83) stipulates that each node has always at least one and at most D_{max} ambassadors in the network with $N \geq \Delta$ nodes, and (iii) According to the node's state, each node is always connected to a head node.

Let's P be the proposition such that P is true when:
"For $N \geq \Delta$, there is a node x that is disconnected from the rest" is true. The proposition P can be expressed in three sub-propositions P_1, P_2, and P_3 as described below, such that P is true, if and only if the sub-propositions are all true:

P_1: x has no head node;

P_2: x has neither a neighbor on the inner, nor on the outer ring;

P_3: x has no De Bruijn neighbors;

Thus, it is sufficient to show that one of the propositions P_1, P_2 or P_3 is not true for $N \geq \Delta$ to demonstrate that P is false.

Let us assume that the sub-proposition P_1 is true. In a network with at least $N \geq \Delta$ nodes, the logical concentric multi-ring topology is formed of at least one fully constructed ring: the innermost ring. If the node x has no head node, then the node x is the ring contact node on the innermost ring. If $N \geq \Delta$ then by definition x has a at least one De Bruijn neighbor with ID $\Delta - i$ where $i < \Delta$ on the innermost ring. This implies that P_3 is false, when P_1 is true.

In conclusion, P is false and Lemma 6.2.2 is true. ■

The Lemma 6.2.2 implies that if at least one fully constructed ring exists in the network, then each node will always be connected with any other. Hence, as the number of nodes in the network grows, the chance that a lookup fails, is negligible. The Lemma 6.2.3 is a consequence of the Lemma 6.2.2.

Lemma 6.2.3 *The network is k-connected with* $k = N - \Delta + 1$.

Proof: The proof follows from Lemma 6.2.2. Since the network requires at least Δ nodes to ensure full connectivity, the network connectivity will be affected only if a subset of at least $N - \Delta + 1$ nodes are removing collectively. ■

Besides, if there is a targeted attack on the $2\Delta + 2$ entries of the routing table of a node x, definitely, x will be disconnected from other nodes in the system. Therefore, the network will be k-connected with $k = 2\Delta + 2$. However, it is improbable that such a situation occurs, because of the following reasons. First, the routing table of a node is dynamically maintained. Thus it is difficult to precisely attack all the nodes of the routing table of a given node x. Second, deleting all the $2\Delta + 2$ links to/from a node x is not easy because the entries of the routing table of a node x are widely distributed geographically.

The Lemma 6.2.3 in other words means that when k nodes are collectively deleted from the network such that $k \leq \frac{N}{2}$, the network can be reconstructed and the consistent state can be re-established.

6.3 Shortest Path to Requested Resource

The discussion in the last section confirms that the network is always in a consistent state with high probability. Hence, the lookup distance for any successful resource lookup operation in the CMR network is $D_{CMR} = \log_{\Delta}(N_{max}(\Delta - 1) + \Delta) - 1$ as presented in Chapter 4. We show here that D_{CMR} is closest to the Moore bound than other outstanding P2P solutions. We state the following:

Theorem 6.3.1 *Let $B(\Delta, D)$ be a De Bruijn digraph with N nodes, where $N = \Delta^D$ and $2 \leq \Delta < N$. One can construct an optimal logical overlay network, where each node has at most $2\Delta + 2$ neighbors and each resource is located in at most $D_{CMR} = \log_\Delta(N_{max}(\Delta - 1) + \Delta) - 1$ overlay hop such that $D_{moore} < D_{CMR} \leq D_{DB}$. D_{moore} being the diameter of a possible Moore graph with the same number of nodes and $D_{DB} = \log_\Delta N$ the diameter of the $B(\Delta, D)$ De Bruijn digraph.*

Proof: First, it has been proved in Lemma 4.7.1 that the lookup of a resource in the network can be achieved in at most $D_{CMR} = \log_\Delta(N_{max}(\Delta - 1) + \Delta) - 1$ hops. Hence, what needs to be proved here is the inequality $D_{moore} < D_{CMR} \leq D_{DB}$ for $2 \leq \Delta < N$. For that purpose, we use the induction proof.
Let us consider the proposition

$$P : \log_\Delta(N(\Delta - 1) + \Delta) - 1 \leq \log_\Delta N$$

such that P is true for $2 \leq \Delta < N$.
For $\Delta = 2$, the following inequalities are true:

$$\begin{aligned} \log_2(N + 2) - 1 &\leq \log_2 2N - 1 \\ \log_2(N + 2) - 1 &\leq \log_2 N + \log_2 2 - 1 \\ \log_2(N + 2) - 1 &\leq \log_2 N \end{aligned}$$

Hence, P is true for $\Delta = 2$.

For $2 \leq \Delta < N$, we can set the following equations:

$$\begin{aligned}
N &\geq \Delta + 1 \\
N\Delta + N &\geq N\Delta + \Delta + 1 \\
N\Delta + \Delta + 1 &\leq N\Delta + N \\
\frac{N\Delta + \Delta + 1}{\Delta + 1} &\leq N \\
\log_{\Delta+1} \frac{N\Delta + \Delta + 1}{\Delta + 1} &\leq \log_{\Delta+1} N \\
\log_{\Delta+1}(N\Delta + \Delta + 1) - \log_{\Delta+1}(\Delta + 1) &\leq \log_{\Delta+1} N \\
\log_{\Delta+1}(N\Delta + \Delta + 1) - 1 &\leq \log_{\Delta+1} N \\
\log_{\Delta+1}[N((\Delta + 1) - 1) + (\Delta + 1)] - 1 &\leq \log_{\Delta+1} N \qquad (6.1)
\end{aligned}$$

From the inequality (6.1), it can be deduced that the proposition P is true for $\Delta + 1$, which means by induction that P is true for all $2 \leq \Delta < N$. ■

According to this theorem, it can be concluded that for any resource lookup operation in CMR, the nearest resource on the overlay is always found within the shortest path. Moreover, the resource lookup path has a length that is closer to the Moore bound that the related work such as Chord, Pastry and Tapestry.

6.4 Generalization

The theoretical analysis proposed here shows that with high probability a lookup can fail only when all the ambassadors of a resource concurrently fail. Beside this worst case scenario, a resource is always found in the system. Furthermore, we show that the lookup of a resource can always be achieved within at most $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$ overlay hops, which is the lowest lookup bound

so far achieved in comparison to the results presented in the related work: in general around $\log_{\Delta} N$. We demonstrate that this lookup bound is closer to the Moore bound than the lookup path length achieved by the other outstanding resource location strategies developed recently.

Chapter 7

Experimental Evaluation of HiPeer

IN this chapter, we will present and discuss the results of the simulation that has been performed to analyze the behavior of HiPeer. For the simulation, we have used a prototype Java implementation of the proposed architecture. Some relevant network parameters have been measured during a given time-period and in different network situations. We measured the path length of resource request messages, the load distribution at the nodes, the routing table size, the variation of the failure detection time, the failure recovery time, and the mean duration of the join procedure. These parameters are presented and discussed in both static network without failures and dynamic network with high level of churns. The intention of this experimental evaluation is to verify the results of the theoretical analysis in the precedent chapters.

7.1 Simulation Settings

For the experimental evaluation of HiPeer, we have simulated a P2P network of about 1300 nodes distributed on seven SUN Fire V210 machines with each $2 \times 1,0$ GHz UltraSPARC III 64 Bits with 2GB RAM of memories, interconnected over a Gigabit Ethernet. The nodes in the simulation environment are managed using their IDs, such that using the same IP-Address will not have any impact.

When inserted in the system, new nodes are launched on a machine of the computer environment that has sufficient processing resources. When a machine can not support creation of new threads for example, another machine is chosen to host the new created node.

During the simulation execution, a node or a set of nodes can join or leave at any time according to a succession of Poisson distributed events as presented in Section 6.1 in the precedent chapter. Each node in the network regularly requests a resource that is well-known in the system. In order to simulate ungraceful node departures in the system, we consider good nodes, — normal nodes, participating in the network to request and distribute resources,— and malicious or adversarial nodes, participating in the network to undermine its performance. The malicious or adversarial nodes are configured such that they can leave the network without any prior warning.

In what follows, we subdivide the simulation execution in two parts. Firstly, we consider a growing network without any failures. This means that, nodes can join continually, while no node is leaving the network. Secondly, we run a simulation, where there are nodes joining and leaving the network in a random fashion. Hence, the description of the simulation and the interpretation of the results are subdivided in two sections. For simplicity reasons, we choose $\Delta = 3$ for the degree of the De Bruijn digraph forming the rings. However, the parameter

Δ can also take other values such as $2, 4, 5, 6, ..., \log N$.

7.2 Observation without Failures

During the first experimental execution of the simulation, nodes have been inserted in the network in a memoryless and stochastic manner, obeying to the Poisson distribution as described in the dynamic model in Section 6.1. In this first phase, the case of nodes leaving the network is not considered, and the network grows to reach a maximum of $N = 1300$ nodes. According to the CMR structure, 1300 nodes can be represented on seven rings, where six of them are fully constructed and the outermost one consists of 208 nodes.

During the network construction phase, each node sends a request for an arbitrary resource at a rate of one request per 15 seconds. The experiment is stopped a few minutes after the expected number of nodes have joined the network and the results are described in the following subsections.

7.2.1 The Lookup Path Length

Here, we have measured the message path length of every resource request operation initiated by nodes in the network. The path length defines the number of peers that a request message traverses before it reaches a valid resource provider or a valid ambassador of the requested resource in the network. The path length of a lookup message is evaluated as a function of the number of nodes present in the network at a given time.

After having made the simulation, we can observe that each requested resource could be found with a probability of 100% in at most $D_{CMR} = \log_{\Delta}(N_{max}(\Delta - 1) + \Delta) - 1$ overlay hops. Figure 7.1 shows that the maximum number of hops for each lookup operation is at most D_{CMR}. In other words, when all the rings

in the network are fully constructed, the 99^{th} percentile of the path length of the resource request messages is bounded at D_{CMR}; this has been demonstrated in Lemma 4.7.1. The 99^{th} percentile of the path length defines the maximum path length which is achieved for each value of the number of nodes in the system. It is represented on Figure 7.1 by the tail of the arrow. On the other hand, the 1^{st} percentile of the path length defines the minimum path length which is achieved for each value of the number of nodes in the system. It is represented on Figure 7.1 by the head of the arrow. The 50^{th} percentile defines the mean. Figure 7.1 also

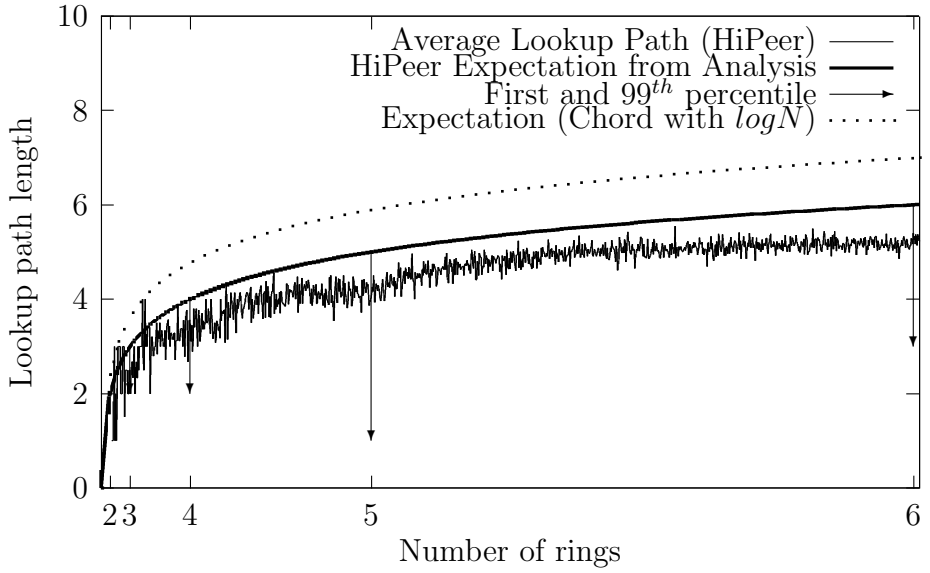

Figure 7.1: The 1^{st} and 99^{th} percentile of the lookup path length in a CMR network of degree $2\Delta + 2$ with $\Delta = 3$ as a function of the number of nodes in the network.

presents the expected path length of a lookup operation in a Chord network with $\log_2 N$ overlay hops comparing to the bound in CMR. We additionally present in Figure 7.2, the average path length for the resource lookup operation. The observations of the simulation's analysis confirm the results of the theoretical analysis. The lookup of any resource in a widely distributed and self-organized system based on the CMR concept can be achieved within a lowest bound.

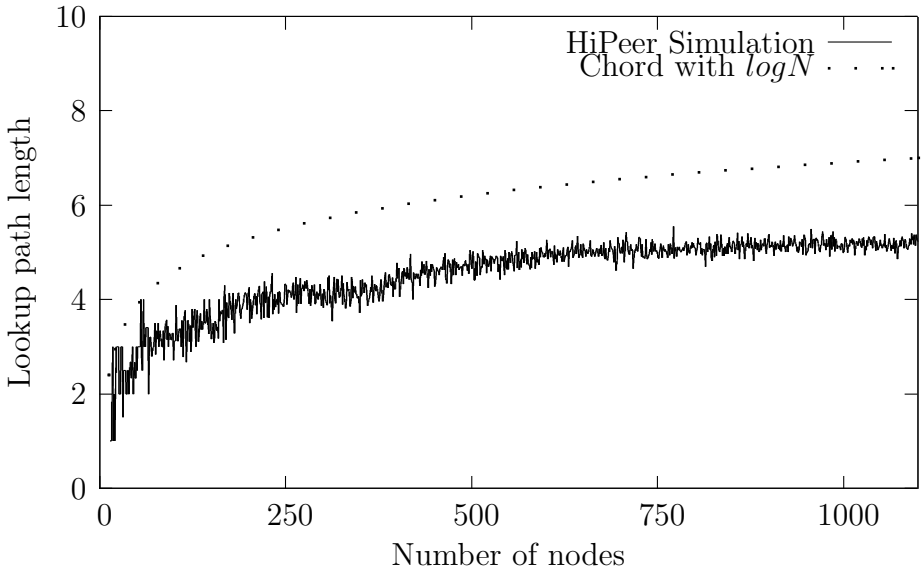

Figure 7.2: The average lookup path length in a CMR-based network with degree $\Delta = 3$ as a function of the number of nodes in the network.

7.2.2 The Load Distribution

In our simulation, we investigated the ability of the resource distribution strategy to allocate keys to nodes evenly. We studied the load balancing of the resource location strategy by measuring the number of resources distributed at each node in the large-scale network at a given time. The P2P network concept is built on the principles of equality between peers in the system. Hence, in the ideal case every node must maintain a well-balanced number of resources and must be involved in an equal number of forwarding operations. In our simulation, we consider a growing network, where each node provides exactly one resource. The network can also be designed to support more than one resource per node.

Figure 7.3 plots the mean and the 99^{th} percentile of the number of keys per node as a function of the total number of resources in the system. As expected, the average number of keys per node exhibits small variations that tends to a constant value as the number of keys is growing. The number of keys stored by each node in this case is limited to the maximum value of 30. The variance of

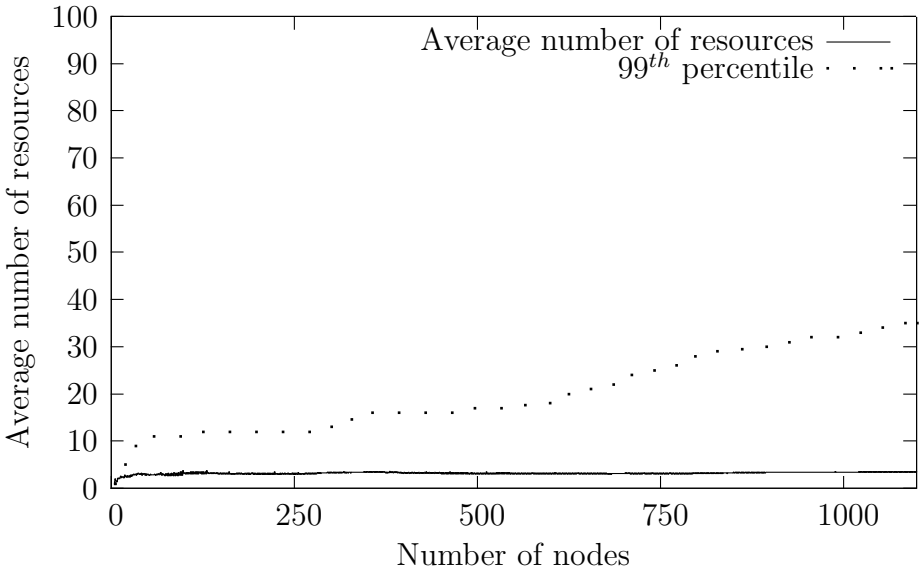

Figure 7.3: The mean and the 99^{th} percentile of the number of resources stored per node as a function of the number of resources in the system. The total number of keys is 1300.

the resource distribution during gradually network construction is equal to

$$\sigma^2 = \frac{1}{N_{max}} \sum_{N_{max}}^{n=0} (\mu - k)^2 = 8.11$$

with the parameter $\mu = 3,29$ being the mean number of resources available per node and k the number of resources available at a node at a given time. The standard deviation to the mean number of resources per node in the network is $\sigma = 2,84$, which indicates that the difference value for the distribution of resources between any two nodes in the network is balanced around ± 3 resources. The small variation of the average function can be explained by the fact that, when new nodes join the network with some new resources, each resource is published to a predefined number of nodes: one on the next inner ring and one on the same ring. In other words, the nodes that are elder in the system can not be chosen as ambassadors for new resources.

However, the 99^{th} percentile representation of Figure 7.3 linearly grows with

the number of keys in the system. This growing tendency is a consequence of the replication strategy included to support range queries. When a new node joins the network on the outermost ring, it becomes responsible for some resources on the same ring and also for a set of resources coming from the underlying rings. This explains why some new joining nodes have more responsibilities than the elder ones in the inner rings. Nevertheless, the nodes in the inner rings maintain elder and stable resources: following the motto "the elder, the trustworthy". This is an interesting property of the node management strategy in the CMR overlay, because the node's insertion procedure has only local consequences. To reach the same capabilities exhibited here, protocols such as Chord or CAN reconfigure the routing tables of their nodes each time a new node joins the network. This procedure creates a huge management overhead and affects a large portion of the network, which is costly and undesirable.

7.2.3 The Routing Table Size

As the network grows, we measured the size of the routing table at each node. Figure 7.4 shows as expected that the routing table size of each node is limited to a maximum of $2\Delta + 2$ and can not grow larger than that. Additionally, a new node insertion in the system does not require any reorganization or shift in the routing table of the already existing nodes. The maintenance cost of the routing table and its size are not dependent on the size of the network. The fact that the routing table size is small and limited to a constant value is a desirable performance property. A small-sized routing table is used to speed up the forwarding decision at each node and even when the nodes regularly trace their entire routing tables, the cost for network maintenance is acceptable. Even better, when detecting failure on the CMR overlay, each node regularly sends only

one KEEPALIVE message to its head node. According to the structure of the

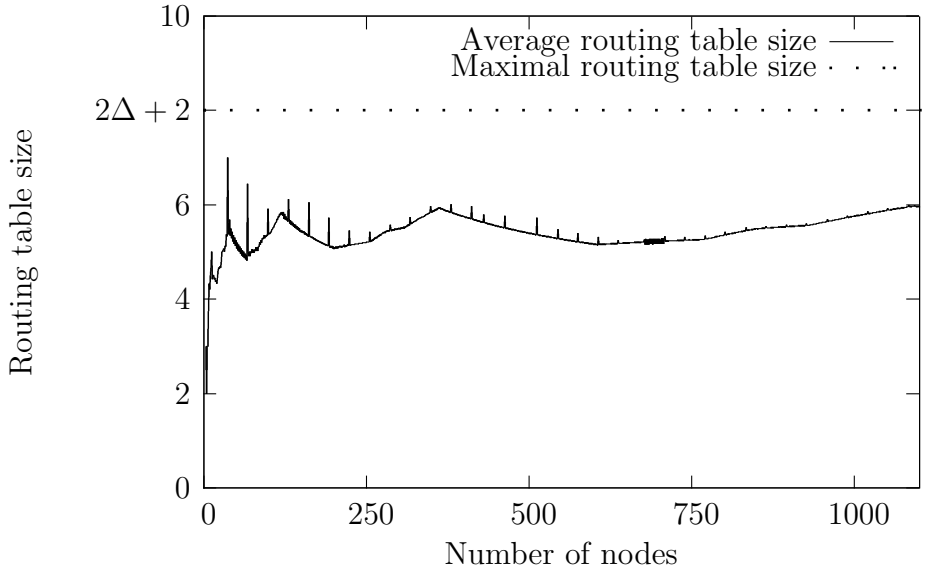

Figure 7.4: The average number, the 1^{st} and 99^{th} percentiles of the routing table entries per node as a function of the network size.

concentric multi-ring overlay, nodes can route only to nodes on the neighboring rings. Hence, even if the network grows to millions or even billions of nodes the routing table remains upper-bounded and each message can always be optimally routed to its destination.

7.2.4 The Maintenance Overhead

After constructing the concentric multi-ring overlay, a neighborhood-awareness strategy is required to ensure that the structure is still in a consistent state. Our maintenance approach requires only one periodic control overhead per node to ensure neighborhood-awareness. Each node joining the network needs to send only one *join_request* message to the network to be inserted correctly and D_{max} nodes are involved in the join procedure to forward the request. Similarly, each node leaving the network causes the generation of $O(\Delta)$ control messages. Both complexities are independent of the number of nodes in the system.

7.2.5 The Duration of the Join Procedure

We have mentioned in the node's insertion procedure of Chapter 5 that new nodes always join the overlay on the outermost ring. Hence, a *join_request* message is always forwarded in the outermost ring to a head node that sends a *join_invite* message to the joining node. Here, we studied the issue of join by measuring the time elapsed between the sending of a *join_request* message and the reception of a *join_invite* message. As presented in Figure 7.5, the join duration time is lower than fifty milliseconds. However, many nodes could wish to join at the same time, sending many *join_request* messages concurrently to the network's contact node. The curve of Figure 7.5 shows some peak values reaching the hundred of milliseconds order, which correspond to the join duration time during a concurrent join procedure as expected.

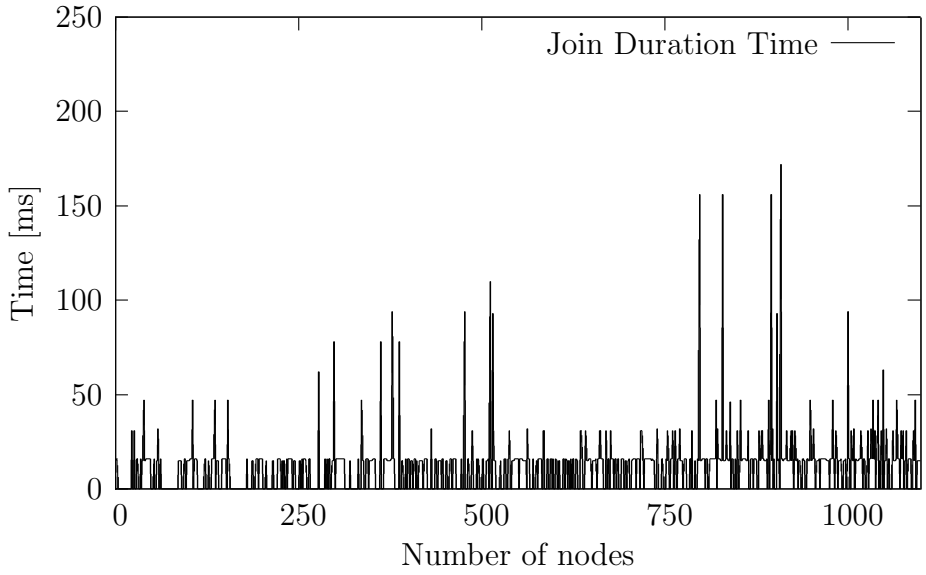

Figure 7.5: The duration of the node joining procedure

7.3 Observation under Failures

After the first observations without network failures, this section will focus on the simulation of a near real world scenario, with nodes joining or leaving individually or collectively. In this second part of the experimental analysis, the node's join and leave event are modeled as a succession of Poisson distributed events. For the execution of the simulation, we will first let the network grow to at least five nodes before starting with the churn process itself. During the simulation, a set of nodes calculated with a Poisson probability function is chosen every ten seconds to join or to leave the network. Thereby, we only make sure that the distribution of the number of nodes in the network is an increasing function. We stop the simulation, after the network has reached a number of $N = 1300$ nodes in the system. This means that the network is formed of 6 fully constructed rings and one incomplete outermost ring with identifier 7 for $\Delta = 3$. To keep the nodes informed about their neighbors, each node sends one KEEPALIVE message to its head node at a rate of one every ten seconds (1 KEEPALIVE per 10 seconds) and requests one resource every fifteen seconds.

During the simulation, we have measured the lookup path length of each resource request operation. We have also measured the routing table size per node, when the number of nodes in the network is growing when other nodes are leaving. We have collected data about the number of resources that are maintained by each node to determine the load distribution of HiPeer in highly dynamic network situations.

7.3.1 The Lookup Path Length

The lookup operation during churns does reveal only some slightly deviation to the simulation's results of the failure-disabled scenario. We found out in the

first experiment that there is no lookup failure as long as there is no node's failure in the network. At a high rate of churns after all, at least 99% of the 194109 requested resources in the network could be successfully found by the requesters. The lookup failure occurs for example, when a node on the route to the destination has been silently deleted from the network and a pointer to its address is still available in the neighbor's routing table. This could be resolved by introducing the notion of acknowledgement for any receiving messages by every node in the network. This is good for the accuracy of failure detection but it may involve some additional control overhead in the resource lookup and in the message routing operations.

Figure 7.6 shows that any requested resource is found within a path length $D_{CMR} = \log_{\Delta}(N(\Delta - 1) + \Delta) - 1$. In Figure 7.7 we show a comparison result of the average path length with the path length in the Chord protocol [113]. The results of this network emulation show that any resource in HiPeer, — even at high rate of churn,— can be reached within a path length of at most D_{CMR} overlay hops, which as close as possible to the Moore bound.

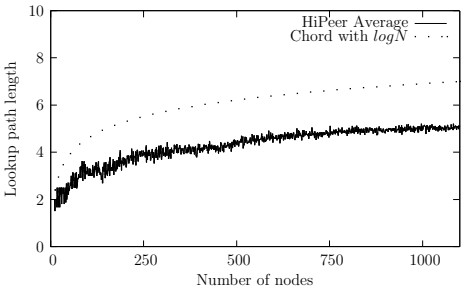

Figure 7.6: The lookup path length in HiPeer with degree $\Delta = 3$ in a network with six fully constructed rings as a function of the number of nodes.

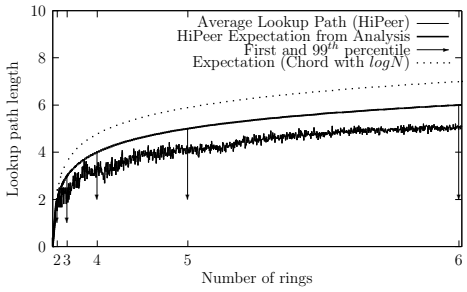

Figure 7.7: The average lookup path length in a gradually growing network as a function of the number of rings in the network.

7.3.2 The Load Distribution

When a node in the inner rings of the overlay leaves the network, some other nodes of the outermost ring change their position to replace the failed node. Moreover for optimization purposes, when a node changes its original ID and its ring, it does not need to delete the resource's pointers from the old location in its location table. It changes the ring while maintaining the original resources and appends the new resources to the later ones. This explains why the variance of the number of resources per node is larger during churns than observed without failures: $\sigma^2 = 45.85$ with a standard deviation of $\sigma = 6,77$. The average number of resources per node however, has a course that is nearly constant.

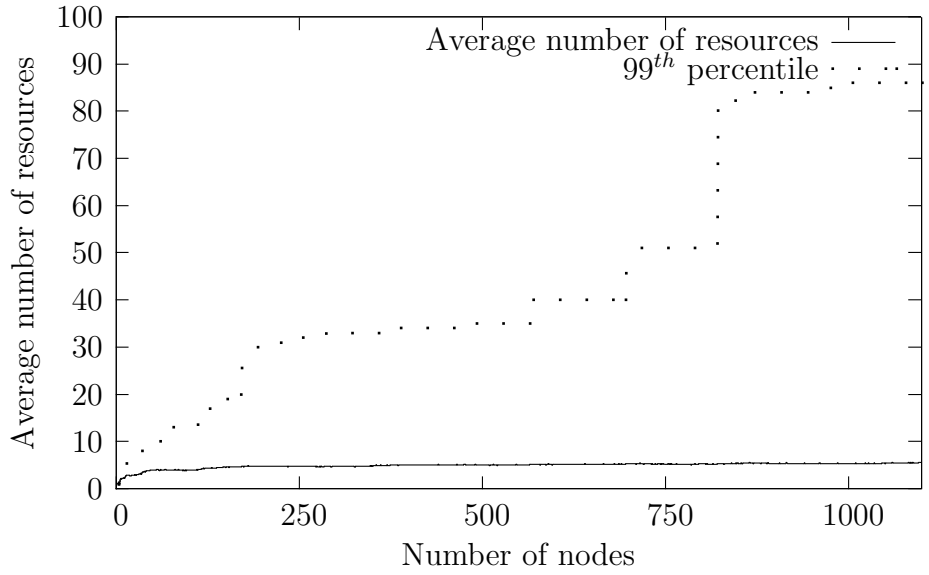

Figure 7.8: The mean and the 99^{th} percentile of the number of resources stored at each node in a failure-prone network as a function of the number of resources.

7.3.3 The Routing Table Size

Generally, in order to enable a good connectivity between the nodes of a network, an intuitive approach is to connect each node with every other in the network.

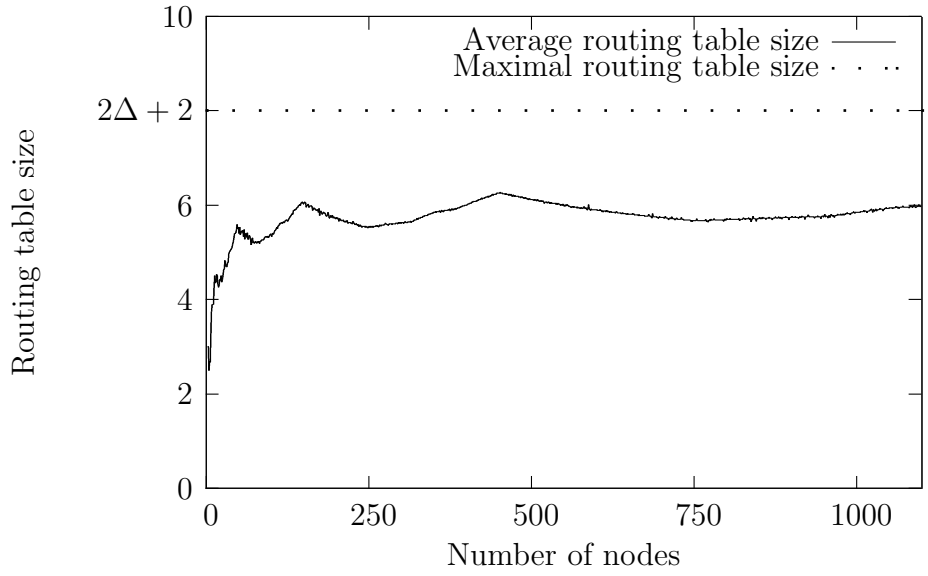

Figure 7.9: The average number of routing table entries per node as a function of the network size.

But, by using such an approach the routing table size of each node in the network will grow to N, which is undesirable. The routing table in related protocols such as Chord [113] and Pastry [99] grows logarithmically with the number of nodes in the network. In HiPeer, however, the routing table is limited to $2\Delta + 2$ independently of the number of nodes in the network and independently of the churn rate. Figure 7.9 shows the evolution of the mean routing table size per node in a network, which is similar to the variation of the routing table in the first experiment.

7.3.4 Failure Detection and Recovery Delay

As mentioned above, the simulation environment is configured such that any node sends a KEEPALIVE message to its head node every $T = 10$ seconds. The graph of Figure 7.10 shows that the failure detection time mainly depends on the choice of the period T. The failure detection time varies in the interval $[0, 10000\ ms]$, because some nodes will fail shortly before the timeout or shortly

after it. The graph of Figure 7.10 also depicts the failure recovery delay which is the time spent from the moment a failure have been detected until the moment the failed node has been replaced. We observe that the failure recovery time is very short depending on whether the failed node and its successor are on the same machine of the simulation platform or not. This can be used to confirm that w.h.p. the network is always in a consistent state because the time τ spent for failure recovery is extremely small and the probability of lookup's failure is $\frac{\tau}{N} \approx 0$ for a large number of nodes N in the system.

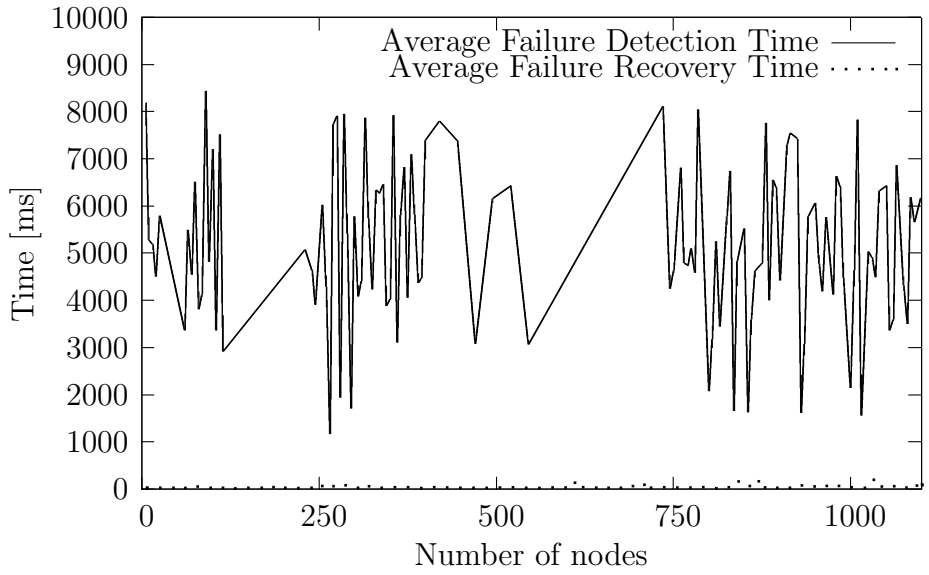

Figure 7.10: The average failure detection and the average recovery time in a highly dynamic network environment.

7.3.5 The Duration of the Join Procedure

The duration of the join procedure has a similar behavior as in the failure-disabled scenario. The course of the join duration time is presented in Figure 7.11. There is no *join_request* failure and no ID duplication. The consequence is that some join processes take more time than other. Nevertheless, a long duration time has no direct impact of the performance of the P2P systems itself.

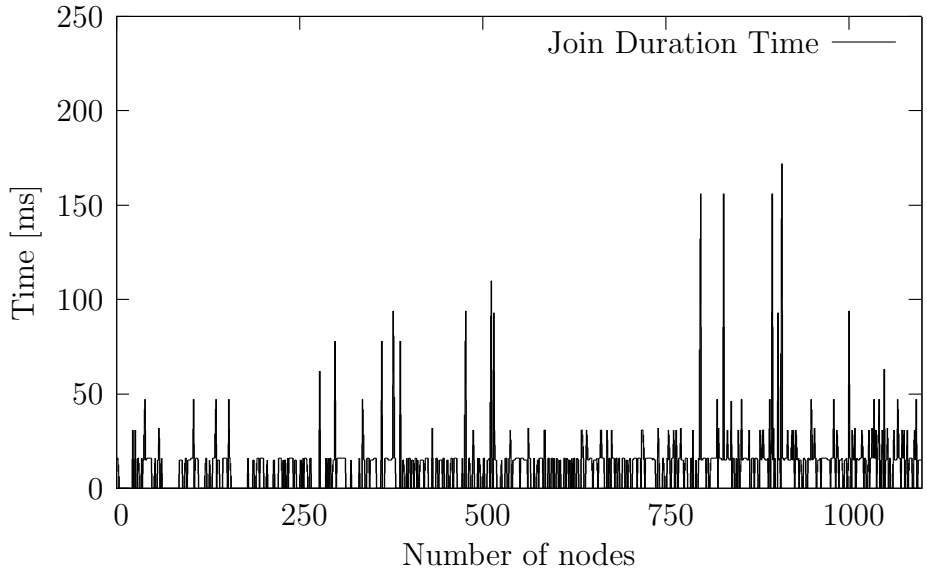

Figure 7.11: The duration of the node's joining procedure

7.4 Summary

In this chapter, we have evaluated through simulation the performance of a Java-based implementation of the proposed resource location strategy. We were able to verify and confirm the values obtained through theoretical analysis from the previous chapters. The simulation has been done in a mostly realistic environment. However, congestion on the communication links and packet lost in the network have not been considered.

Chapter 8

Conclusion

THe objective of this thesis has been to propose an appropriate resource management solution in P2P systems. In this chapter, we firstly summarize what has been achieved within the scope of this dissertation; secondly, we conclude by comparing the thesis' achievements against the requirements defined in Chapter 1 and then we give an outlook to future research.

8.1 Summary of the Contributions

Within the scope of this thesis, we have proposed HiPeer, an evolutionary solution to the problem of resource management in P2P systems. The system model is based on rings like in the Chord protocol [113]. In contrary to the single ring-based solution of Chord, HiPeer is based on a concentric multi-ring (CMR) overlay that is used for the representation of nodes or resources in a structured virtual network. The nodes within the CMR overlay are organized such that each node in the system maintains a list of at most $2\Delta + 2$ neighbors widely distributed in the system. Each node distributes its resources to at most D_{CMR} other nodes in the network using an extended version of the distributed hash table. Each

requested resource is found with a high probability of over 99% within at most $D_{CMR} = \log_{\Delta}(N_{max}(\Delta - 1) + \Delta) - 1$ overlay hops.

We have shown in this thesis that our approach enables the construction of a highly reliable churn-resistant architecture for P2P networks. We have simulated the proposed approach and we have compared it to the Chord protocol to demonstrate that HiPeer performs better. Although other structured approaches to P2P networking such as Pastry, Tapestry and Kademlia are also stable, they present some performance limitations such as the high cost for network management during nodes' join, leave and network update $O(\log N)$, as well as large routing table per node with a complexity of $O(\log N)$.

The proposed solution exploits the interesting properties of De Bruijn digraph and the superposition of rings in the multi-ring architecture (flexible routing options). HiPeer demonstrates some interesting performance results. It was shown that the maximal lookup distance D_{CMR} between a resource requester and a resource provider or an ambassador is the lowest bound achieved so far in P2P systems. We have studied the influence of churns on the construction and the maintenance of the network overlay and we have shown that it is possible to maintain the overlay structure in a consistent state using the lowest possible cost of one control overhead per node. The network management operations during node's insertion or departure from the network can be achieved with an optimal constant cost of $O(\Delta)$. We've shown that even when half of the nodes in the network, — $\frac{N}{2}$ nodes,— leave concurrently, the network can be successfully reconstructed with high probability and each request can be answered within at most D_{CMR} overlay hops. Moreover, when considering only nodes joining the network, any resource lookup succeeds with a high probability of 100%.

8.2 HiPeer vs. Requirements for Resource Management in P2P Systems

Here, we summarize the properties of HiPeer with regard to the requirements list of Section 1.2. Table 8.1 gives an overview of the degree of matching.

Fault-tolerance: Node's insertion and departure in HiPeer are reactively handled so that the network is always in a consistent state, (refer to Chapter 5). We have shown that even in the worst case, when half of the nodes in the network fail simultaneously, routing and lookup operation in the network can be successfully completed with a probability of over 99%, (refer to Chapter 6). Hence, HiPeer is designed to support efficient operations even during nodes' failures as proved in the Theorem 6.2.1 on Page 117.

Shortest path for lookup operation: Each lookup query has a maximal path of length lower than D_{CMR}. We have shown through theoretical analysis (in Chapter 6) and simulation (in Chapter 7) that the lookup path of length D_{CMR} for $2\Delta + 2$ neighbors per node is closer to the Moore Bound than any other outstanding P2P solution such as Chord, Pastry or Tapestry. This has been proved in the Theorem 5.4.1 on Page 111 and the Theorem 6.3.1 on Page 120.

Low network maintenance cost: The operation of node's insertion and the network stabilization after a node's failure in the HiPeer network is achieved with a cost of $2\Delta + 2$ nodes involved. In order to detect failure and keep the network up-to-date each node in HiPeer sends no more than 1 periodic control message to its head node, refer to Section 5.3 for details.

High Availability: We used simulation in Chapter 7 to confirm the results of the theoretical analysis that at least 99% of the requests in HiPeer can be successfully answered when the requested resource exists.

However, these interesting results were not achieved without cost. The tasks for network management and information storage are indeed fairly distributed among nodes in the system, but some nodes in the concentric multi-ring overlay save more information about the network than others as discussed in Sections 7.2.2 and 7.3.2. This depends on their position on the overlay, where nodes on the inner rings have larger location tables than nodes on the outer rings. Nevertheless, this problem is not critical, because the variation of the task distribution is small with a standard deviation of $\sigma = 6,77$ in a highly dynamic network and $\sigma = 2,84$ in network with joining nodes only.

8.3 Future Research

A foundation for scalable and robust P2P architectures has been proposed in this thesis. The P2P concept in general enables direct and cost effective deployment of communication networks without the need of a central management instance. It allows communication without boundaries and with management cost well-balanced among network participants. The P2P concept is a promising enabling technology for computer systems relying on self-organization principles.

We argue that the approach of P2Ps proposed in this thesis deserves closer attention. It can be extended to address other interesting aspects of P2P research such as security, trust, anonymity and locality-awareness. Moreover, the proposed CMR architecture can be used for the design of fault-tolerant systems in areas such as reconfigurable and dependable computing systems. In what follows, we present some future research directions and discuss their significance.

Protocol Class/Requirements	**Fault-tolerance**	**Lookup Path Length**	**Cost for Maintenance**	**Availability**	**Load Balancing**
Unstructured P2Ps	Yes	$O(N)$	Low	Low	Poor
Flat DHT P2Ps	Yes	$\log N$	variable ($\log N$)	High	Good
Hierarchical DHT P2Ps	Yes	Dependent on the inter group management strategy	Dependent on the inter group management strategy	Dependent on the inter group management strategy	Poor
Non-DHT P2Ps	Yes	$\log N$	High $O(\log N)$	Middle	Poor
Loosely Structured P2Ps	Yes	$\log N$	Low	Middle	Poor
HiPeer	Yes	D_{CMR}	$O(\Delta)$	High	Middle

Note that $D_{CMR} = \log_{\Delta}(N_{max}(\Delta - 1) + \Delta) - 1$

Table 8.1: HiPeer vs. the related work with regard to the requirements for P2Ps.

8.3.1 Locality-awareness

Depending on the type of P2P application being used, one will prefer to locate resources that are next in the geographic proximity to the resource requester. This means that the nodes should be organized such that they are aware of each others locality. The concentric multi-ring architecture does not explicitly address the issue of geographic proximity, but it can easily be extended to fulfill that purpose.

Some proposals of locality-aware or proximity-aware P2Ps are given in the literature such as HIERAS [121], Cyclone [23], Grapes [110] and Canon [49].

However, they are mostly limited because they require geo-positioning-enabling devices such as GPS [73] or the existence of fixed landmarks or well-defined locality nodes [121, 110]. One promising future research direction is the development of a proximity-aware concentric multi-ring overlay network for improving the CMR architecture and for supporting applications requiring proximity-awareness. Such an algorithm could rely for instance on the hierarchical P2P network concept by extending the ideas proposed in more generic approaches such as in HIERAS [121] or Canon in G. Major [49].

Another interesting research topic that goes in the same direction is to design the interconnection between two or more CMR overlays representing different organizations, cities, countries or continents.

8.3.2 A Semantic-based CMR Overlay

Besides locality-awareness, some special applications, such as those for management of heterogeneous resources in a grid, will need semantic-oriented virtual organization of resources. The nodes of a grid are of heterogeneous nature and their availability can vary during the lifetime of the network. In such environments, it is of importance to keep and publish an always fresh view of the nodes in order to ensure content-oriented filtering and semantic-oriented search or routing.

The CMR architecture presented in this thesis is a generic and scalable architecture that can be adapted for the semantic organization of nodes and resources of a large-scale network. Hence, a possible future work will be to investigate the development of a semantic-based P2P overlay based on the CMR. The semantic-based overlay can be applied to domains such as P2P-Grid or network operating systems such as ISOS [20] and other aspects of autonomic systems.

8.3.3 Concentric Multi-spherical Architecture

One question of interest is whether it is feasible, according to flexibility and efficiency, to extend the CMR to a type 3D gyroscope-like concentric multi-ring or even a concentric multi-spherical (CMS) architecture. In other words, we are thinking of replacing the rings with spheres to obtain a CMS overlay. Indeed, within a concentric multi-spherical overlay, the network is organized such that the nodes have more neighbors on different overlay planes, that enables better robustness and more routing options during faults. However, it is also worth mentioning that a multi-spherical architecture will require a much more complex management solution than rings and would generate more control messages to keep the larger list of neighbors up-to-date. Nevertheless, we believe that this could be an interesting extension that could unveil beneficial properties for an on the fly construction of virtual structures.

8.3.4 The CMR as a Scalable Design Architecture for Dependable Computing

One of the main research topics in the area of reconfigurable and dependable computing [28, 57] is the design of fault-tolerant architectures for enabling efficient routing and flexible interaction between the components of the systems (e.g. interaction between the gates of a FPGA [29] and distributed storage systems [111]).

The concentric multi-ring architecture has a flexible and extensible design that enables the easy interconnection of a large number of components or nodes within a highly reliable network. Thus, CMR represents an attractive design approach to be used as a basic for the design of fault-tolerant architectures. Hence, a future work is to investigate whether it is worthy to extend the CMR design for

dependable and reconfigurable computing.

8.3.5 Large-scale Deployment of HiPeer

In this work, we have implemented a Java version of HiPeer with the concentric multi-ring organization of the nodes in the network. This implementation has been used for the simulation of the CMR overlay as we have shown in Chapter 7. One of the intentions is to develop prototype P2P applications based on HiPeer that would be used for resource organization in the Internet or that could be used to allow accessibility of resources in grid computing systems. An Internet-based P2P applications will be helpful to stress the protocol. Besides, with the large scale deployment, some undetected problems or limitations, if they exist, would be brought to light. Moreover, P2P development and deployment platforms such as Jxta [4] could be used as benchmarking environment.

Bibliography

[1] Freepastry's homepage. http://freepastry.rice.edu/.

[2] Grokster official homepage. http://www.grokster.com.

[3] imesh official homepage. http://www.imesh.com.

[4] Jxta's homepage. http://www.jxta.org.

[5] Kazaa official homepage. http://www.kazaa.com.

[6] Morpheus official homepage. http://www.morpheus.com.

[7] Official bluetooth's homepage. http://www.bluetooth.com.

[8] Official gnutella's homepage. http://www.gnutella.com.

[9] Official napster's homepage. http://www.napster.com.

[10] Open ldap website. http://www.openldap.org.

[11] Seti@home's official website. http://setiathome.ssl.berkeley.edu/.

[12] Windows internet naming service. http://searchwin2000.techtarget.com/sDefinition/0,,sid1_gci214128,00.ht%ml.

[13] Jini architectural overview january 1999. http://www.jini.org, January 1999. Sun White Paper.

[14] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[15] I. Abraham, D. Malkhi, and O. Dobzinski. Land: stretch $(1 + \epsilon)$ locality-aware networks for dhts. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 550–559. Society for Industrial and Applied Mathematics, 2004.

[16] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power law networks. In *In Phys. Rev. E64*, pages 46135–46143,, 2001.

[17] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles*, Charleston, SC, December 1999.

[18] S. Albayrak. Agent-oriented technology for telecommunications: introduction. *Commun. ACM*, 44(4):30–33, 2001.

[19] S. Albayrak and R. Sesseler. Service-ware framework for developing 3g mobile services. In *The Sixteenth International Symposium on Computer and Information Sciences, ISCIS XVI*, Germany, 2001.

[20] D. Anderson and J. Kubiatowicz. The worldwide computer. *Scientific American*, 286(3):28–35, March 2002.

[21] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[22] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[23] M. Artigas, P. G. Lopez, J. Ahullo, and A. G. Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *Fifth IEEE International Conference on Peer-to-Peer Computing, 2005. P2P 2005.*, pages 49– 56, 31-02 Aug. 2005.

[24] J. Aspnes and G. Shah. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[25] B. Awerbuch and C. Scheideler. The hyperring: a low-congestion deterministic data structure for distributed environments. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 318–327, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[26] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.

[27] A. Bhargava, K. Kothapalli, C. Riley, C. Scheideler, and M. Thober. Pagoda: a dynamic overlay network for routing, data management, and multicasting. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 170–179. ACM Press, 2004.

[28] W. Brockmann, E. Maehle, and F. Mösch. Organic fault-tolerant control architecture for robotic applications. In 4^{th} *IARP/IEEE-RAS/EURON*

Workshop on Dependable Robots in Human Environments, Nagoya, Japan, June 2005.

[29] S. Brown. Fpga architectural research: A survey. *IEEE Design & Test of Computers*, 13(4):9–15, 1996.

[30] F. Cao and D.-Z. Du. Fault-tolerant routing and multicasting in butterfly networks. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 455–460, New York, NY, USA, 1999. ACM Press.

[31] R. Castaneda and S. Das. Query Localization Techniques for On-demand Routing Protocols in Ad Hoc Networks. *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), Seattle, WA*, pages 186–194, August 1999.

[32] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), Oct. 2002.

[33] Y.-L. Chang and C.-C. Hsu. Routing in wireless/mobile ad-hoc networks via dynamic group construction. *Mob. Netw. Appl.*, 5(1):27–37, 2000.

[34] C. Chen, D. P. Agrawal, and J. R. Burke. dbcube: A new class of hierarchical multiprocessor interconnection networks with area efficient layout. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1332–1344, 1993.

[35] J. W. Chunlin Yang. Dominating-set-based searching in peer-to-peer networks. 3032:332 – 339, Januar 2004.

[36] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed

anonymous information storage and retrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, June 2000.

[37] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol (OLSR).* IETF RFC3626, request for comments: 3626 edition, October 2003.

[38] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 25–30, New York, NY, USA, 2004. ACM Press.

[39] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proc. of the* 22^{nd} *International Conference on Distributed Computing Systems (ICDCS'02)*, page 23. IEEE Computer Society, 2002.

[40] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 15(10):908–920, 2004.

[41] N. de Bruijn. A combinatorial problem. In *Nederl. Akad. Wetensh. Proc. 49*, pages 768–764, 1946.

[42] C. Delorme. Examples of products giving large graphs with given degree and diameter. *Discrete Appl. Math.*, 37-38:157–167, 1992.

[43] Ding-Zhu Du and D. Frank Hsu. *Combinatorial Network Theory*, volume 1 of *1948-III Series*, chapter 3: De Bruijn Digraphs, Kautz Digraphs, and Their Generalizations, pages 65–105. Kluwer Academic Publishers. Printed in the Netherlands, 1996.

[44] R. Dougherty and V. Faber. The Degree-Diameter Problem for Several

Varieties of Cayley Graphs I: The Abelian Case. *SIAM J. Discret. Math.*, 17(3):478–519, 2004.

[45] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[46] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 94–103, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[47] P. Fraigniaud and P. Gauron. Brief announcement: an overview of the content-addressable network d2b. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 151–151. ACM Press, 2003.

[48] A.-T. Gai and L. Viennot. Broose: A practical distributed hashtable based on the de-bruijn topology. In *Peer-to-Peer Computing*, pages 167–164. IEEE Computer Society, 2004.

[49] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *ICDCS '04: Proc. of the* 24^{th} *International Conference on Distributed Computing Systems (ICDCS'04)*, pages 263–272. IEEE Computer Society, 2004.

[50] P. Ganesan and G. S. Manku. Optimal routing in chord. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 176–185. Society for Industrial and Applied Mathematics, 2004.

[51] J. Garcia-Luna-Aceves and M. Spohn. Scalable Link-State Internet Routing. *Proc. of the IEEE International Conference on Network Protocols (ICNP), Austin, TX*, October 1998.

[52] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple service discovery protocol/1/0. Work in progress, ietf internet draft draft-cai-ssdp-v1-03.txt, IETF, October 1999.

[53] J. Gómez, M. A. Fiol, and O. Serra. On large (Δ,D)-graphs. *Discrete Math.*, 114(1-3):219–235, 1993.

[54] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329, New York, NY, USA, 2003. ACM Press.

[55] I. Gupta, K. P. Birman, P. Linga, A. J. Demers, and R. van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, volume 2735 of *Lecture Notes in Computer Science*, pages 160–169. Springer.

[56] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. Rfc 2608, IETF, June 1999.

[57] R. Hagenau, C. Albrecht, E. Maehle, and A. C. Döring. Parallel processing in network processor architectures. In *it - information technology*, pages 123–131. Oldenbourg, München 2005.

[58] S. Hand and T. Roscoe. Mnemosyne: Peer-to-peer steganographic storage. In *IPTPS '01: Revised Papers from the* 1^{st} *International Workshop on Peer-to-Peer Systems*, pages 130–140. Springer-Verlag, 2002.

[59] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[60] N. J. A. Harvey and J. I. Munro. Deterministic skipnet. *Inf. Process. Lett.*, 90(4):205–208, 2004.

[61] A. J. Hoffman and R. R. Singleton. On Moore graphs with diameters 2 and 3. 4:497–504, 1960.

[62] N. Homobono and C. Peyrat. Connectivity of imase and itoh digraphs. *IEEE Trans. Comput.*, 37(11):1459–1461, 1988.

[63] R. Hsiao and S.-D. Wang. Jelly: A dynamic hierarchical p2p overlay network with load balance and locality. In *In the Proc. of the 24th International Conference on Distributed Computing Systems Workshops (ICDCS 2004 Workshops), 23-24 March 2004, Hachioji, Tokyo, Japan*, pages 534–540. IEEE Computer Society, 2004.

[64] M. Imase and M. Itoh. Design to minimize diameter on building-block network. *IEEE Trans. Comput.*, C-30:439–443, June 1981.

[65] M. Imase and M. Itoh. A design for directed graphs with minimum diameter. *IEEE Trans. Comput.*, C-32:782–784, Sept 1983.

[66] J. Li and J. Jannotti and D.S.J. De Couto and D.R. Karger and R. Morris. A Scalable Location Service for Geographic Ad Hoc Routing. *Proc. of the*

ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), Boston, MA, August 2000.

[67] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 661–672. VLDB Endowment, 2005.

[68] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS*, pages 98–107, 2003.

[69] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *CIKM '02: Proceedings of the* 11^{th} *International conference on Information and knowledge management*, pages 300–307. ACM Press, 2002.

[70] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM Press.

[71] B. Karp and H. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Ad Hoc Networks. *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), Boston, MA*, August 2000.

[72] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[73] Y.-B. Ko and N. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), Dallas, TX*, October 1998.

[74] J. Kohl and C. Neuman. *The Kerberos Network Authentication Service (V5)*. IETF, http://www.ietf.org/rfc/rfc1510.txt, September 1993.

[75] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceeedings of the* 9^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[76] A. Kumar, S. Merugu, J. J. Xu, and X. Yu. Ulysses: A robust, low-diameter, low-latency peer-to-peer network. In *ICNP '03: Proceedings of the* 11^{th} *IEEE International Conference on Network Protocols*, page 258. IEEE Computer Society, 2003.

[77] G. Kwon and K. D. Ryu. Bypass: Topology-aware lookup overlay for dht-based p2p file locating services. In *In Proc. of the 10th International Conference on Parallel and Distributed Systems (ICPADS 2004), 7-9 July 2004, Newport Beach, CA, USA*, pages 297–. IEEE Computer Society, 2004.

[78] B. N. L. *Algebraic Graph Theory*. Cambridge Math. Library, 1974, 1993 (2nd edition), 1993.

[79] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242, New York, NY, USA, 2002. ACM Press.

[80] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 395–406. ACM Press, 2003.

[81] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the* 16^{th} *International conference on Supercomputing*, pages 84–95. ACM Press, 2002.

[82] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC*, pages 183–192, 2002.

[83] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[84] E. P. Markatos. Tracing a large-scale peer to peer system: An hour in the life of gnutella. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 65, Washington, DC, USA, 2002. IEEE Computer Society.

[85] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the* 1^{st} *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer-Verlag, 2002.

[86] P. Mockapetris. Domain names - implementation and specification. RFC 1035, November 1987.

[87] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 50–59. ACM Press, 2003.

[88] G. Pandurangan. Building low-diameter p2p networks. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 492, Washington, DC, USA, 2001. IEEE Computer Society.

[89] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Higher Education, 4^{th} edition, 2002.

[90] L. Penserini, L. Liu, J. Mylopoulos, M. Panti, and L. Spalazzi. Cooperation strategies for agent-based p2p systems. *Web Intelli. and Agent Sys.*, 1(1):3–21, 2003.

[91] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers. *Proceedings of the ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications, London, UK*, September 1994.

[92] C. Perkins and E. Royer. Ad Hoc On-Demand Distance Vector (AODV) Routing. *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), New Orleans, LA*, February 1999.

[93] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory Comput. Syst.*, 32(3):241–280, 1999.

[94] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172. ACM Press, 2001.

[95] O. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha. Service discovery in agent-based pervasive computing environments. *Mob. Netw. Appl.*, 9(6):679–692, 2004.

[96] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[97] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM*, 2002.

[98] M. Ripeanu and I. T. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *IPTPS '01: Revised Papers from the* 1^{st} *International Workshop on Peer-to-Peer Systems*, pages 85–93, London, UK, 2002. Springer-Verlag.

[99] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[100] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, Oct. 2001.

[101] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In J. Crowcroft

and M. Hofmann, editors, *Networked Group Communication, 3rd International COST264 Workshop (NGC'2001)*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43, Nov. 2001.

[102] E. Royer and C.-K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Networks. *IEEE Personal Communications*, 6(2):46–55, April 1999.

[103] S. Basagni and I. Chlamtac and V.R. Syrothiuk and B.A. Woodward. A Distance Routing Effect Algorihtm for Mobility. *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), Dallas, TX*, pages 76–84, October 1998.

[104] J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *IPTPS '01: Revised Papers from the 1st International Workshop on Peer-to-Peer Systems*, pages 270–279. Springer-Verlag, 2002.

[105] M. R. Samatham and D. K. Pradhan. The de bruijn multiprocessor network: A versatile parallel processing and sorting network for vlsi. *IEEE Trans. Comput.*, 38(4):567–581, 1989.

[106] S. Saroiu, P. K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *SPIE Multimedia Computing and Networking (MMCN2002)*, 2002. `http://www.cs.washington.edu/homes/gribble/papers/mmcn.pdf`.

[107] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup - hypercubes, ontologies, and efficient search on peer-to-peer networks. In G. Moro and M. Koubarakis, editors, *Agents and Peer-to-Peer Computing*

(AP2PC'02), volume 2530 of *Lecture Notes in Computer Science*, pages 112–124. Springer, July 2002.

[108] H. Shen, C.-Z. Xu, and G. Chen. Cycloid: A constant-degree and lookup-efficient p2p overlay network. In *In Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004*. IEEE Computer Society, 2004.

[109] W. Shi and P. K. Srimani. Hyper-butterfly network: A scalable optimally fault tolerant architecture. In *IPPS '98: Proceedings of the* 12^{th}*. International Parallel Processing Symposium*, pages 732–736, Washington, DC, USA, 1998. IEEE Computer Society.

[110] K. Shin, S. Lee, G. Lim, H. Yoon, and J. S. Ma. Grapes: Topology-based hierarchical virtual network for peer-to-peer lookup services. In *In the Proc. of the 31st International Conference on Parallel Processing Workshops (ICPP 2002 Workshops), 20-23 August 2002, Vancouver, BC, Canada*, pages 159–166. IEEE Computer Society, 2002.

[111] P. Sobe. Reconfiguration of raid-like data layouts in distributed storage systems. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, pages 212–. IEEE Computer Society, 2004.

[112] F. A. M. Specification. http://www.fipa.org/ specs/fipa00023/, 2005.

[113] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, 2001.

[114] J. H. Terpstra and J. R. Vernooij. *The Official Samba-3 HOWTO and Reference Guide*. Prentice Hall PTR, 2004.

[115] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 102. IEEE Computer Society, 2003.

[116] G. Wepiwé and S. Albayrak. A churn-resistant strategy for a highly reliable p2p system. In *Embedded and Ubiquitous Computing - EUC 2005 Workshops, Nagasaki, Japan, December 6-9, 2005, Proceedings*, volume 3823 of *Lecture Notes in Computer Science*, pages 756–765. Springer, 2005.

[117] G. Wepiwé and P. L. Simeonov. A concentric multi-ring overlay for highly reliable p2p systems. In *Network Computing and Applications (NCA05)*, pages 83–90, Cambridge, MA, USA, July 2005. IEEE Computer Society.

[118] G. Wepiwé and P. L. Simeonov. HiPeer: A Highly Reliable P2P System. *IEICE Transactions on Information and Systems*, E89-D(2):570–580, 2006.

[119] R. H. Wouhaybi and A. T. Campbell. Phenix: Supporting resilient low-diameter peer-to-peer topologies. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004. IEEE, 2004*, volume 23, 2004.

[120] K. Xu, X. Hong, and M. Gerla. Landmark routing in ad hoc networks with mobile backbones. *J. Parallel Distrib. Comput.*, 63(2):110–122, 2003.

[121] Z. Xu, R. Min, and Y. Hu. Hieras: A dht based hierarchical p2p routing algorithm. In *In Proc. of the 32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan*, pages 187–, 2003.

[122] D. Xuan, S. Chellappan, and M. Krishnamoorthy. Rchord: An enhanced chord system resilient to routing attacks. In *ICCNMC '03: Proc. of the 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03)*, page 253. IEEE Computer Society, 2003.

[123] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS '02: Proc. of the* 22^{nd} *International Conference on Distributed Computing Systems (ICDCS'02)*, page 5. IEEE Computer Society, 2002.

[124] K. C. Zatloukal and N. J. A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 308–317, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[125] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. *SIGCOMM Comput. Commun. Rev.*, 32(1):79–79, 2002.

[126] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE J-SAC*, 22(1):41–53, January 2004.

[127] Q. Zheng, W. Peng, Y. Wang, and X. Lu. An efficient broadcast algorithm based on connected dominating set in unstructured peer-to-peer network. In X. Zhou, S. Y. W. Su, M. P. Papazoglou, M. E. Orlowska, and K. G. Jeffery, editors, *WISE*, volume 3306 of *Lecture Notes in Computer Science*, pages 724–729. Springer, 2004.

[128] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area

data dissemination. In *NOSSDAV '01: Proceedings of the* 11^{th} *International workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM Press, 2001.

Appendix A

Graph Theory and Moore Bound

THe degree-diameter problem is not a new problem. It has been extensively studied in graph theory and related disciplines, [44, 42, 53] for many years already. In the "degree-diameter problem" one wishes to maximize the number of nodes that can participate in a connected network with a given fixed-degree such that the network diameter is minimal. Here, we define the notion of degree of a vertex $\Delta(x)$ as the number of vertices adjacent to x and the degree of a graph G as $\Delta_G = \max\{\Delta(x), x \in V\}$ with G defined as $G = (V, E)$, where V is the set of vertex and E is the set of Edge of G. The diameter of a graph is defined as the minimum distance between the two most distant nodes in the network. The Moore bound is known as the largest number of nodes that can be connected in a network with a given degree Δ and a given diameter D. The term Moore graph is used in the literature to name a graph which achieves the Moore bound .

The Moore graphs [78] remain the best proposals so far known for constructing the largest connected network with constant degree Δ and diameter D so that

the maximal number of nodes N_{max} in the network is

$$\begin{aligned} N_{max} &= 1 + \Delta \sum_{i=1}^{D} (\Delta - 1)^{i-1} \\ &= \begin{cases} 1 + 2D, & for\ \Delta = 2 \\ \frac{\Delta(\Delta-1)^D - 2}{\Delta - 2} & for\ \Delta > 2 \end{cases} \end{aligned} \tag{A.1}$$

For $\Delta > 2$, this value is called the Moore bound. These are complete graphs, polygon graphs (regular graphs of degree 2) and three others: (nodes, degree, diameter) = (10,3,2), (50,7,2) and the possible but undiscovered (3250,57,2) [78, 61]. The diameter D of a Moore graph as derived from the Moore bound in (A.1) is:

$$D = \log_{\Delta-1}(N_{max}(\Delta - 2) + 2) - \log_{\Delta-1} \Delta \tag{A.2}$$

The problem in P2P systems however is similar to the degree-diameter problem applied to dynamic networks. The question here is how to design a dynamic decentralized self-organized systems such that for a given number of nodes in the system and a given degree, the network diameter is minimal. In P2P systems, when the resource lookup operation within a distance D closest to the Moore bound for a given degree maximal Δ, the lookup is said to be achieved within the shortest path.

Appendix B

Acronyms

ALOHA Area Locations of Hazardous Atmospheres.

AMS Agent Management System.

ARPA Advanced Research Projects Agency.

ARPANET Advanced Research Projects Agency Network.

BFS Breath First Search.

CAN Content Addressable Network.

CCC Cube-Connected-Cycles.

CDS Connected Dominating Set.

CMR Concentric Multi-Ring overlay.

CMS Concentric Multi-Spherical overlay.

CPU Computer Processing Unit.

CTSS Compatible Time-Sharing System.

DA Directory Agent.

DB De Bruijn graph.

DF Directory Facilitator.

DFS Depth First Search.

DHT Distributed Hash Table.

DNS Domain Name System.

DSHT Distributed Sloppy Hash Table.

DSL Digital Subscriber Line.

DynDNS Dynamic Domain Name System.

FIFO Fist-In, Fist-Out.

FIPA Foundation for Intelligent Physical Agents.

FTP File Transfer Protocol.

GPS Global Positioning Systems.

GUID Global Unique IDEntifier (in Chord).

GUIDE Global Unique IDEntifier (in HiPeer).

HiPeer Highly Reliable P2P Systems.

ID Identification number.

IN Intentional Name.

INR Intentional Naming Resolver.

INS Intentional Naming System.

IP Internet Protocol.

ISO International Standard Organization

KEEPALIVE heartbeat message to inform about aliveness.

LAN Local Area Network.

LDAP Lightweight Directory Access Protocol.

LRU Least Recently Used.

MANET Mobile Ad-hoc Network.

MP3 MPEG layer 3.

NETBIOS NETwork Basic Input Output System.

NIS Network Information Service.

ODRI Optimal Diameter Routing Infrastructure.

OMA Object Management Architecture.

OS Operating System.

OSI Open System Interconnection.

P2P Peer-to-peer Systems.

PAN Personal Area Network.

PC Personal Computer.

PING Packet INternet Groper

PRNET Packet Radio Network.

RTT Round Trip Time.

SA Service Agent.

SDP Service Discovery Protocol in Bluetooth.

SLP Service Location Protocol.

SM Salutation Manager.

SSDP Simple Service Discovery Protocol.

SURAN Survivable Adaptive Networks.

TCP Transport Control Protocol.

TTL Time-To-Live.

UA User Agent.

UDP User Datagram Packet.

ULSI Ultra Large Scale Integration.

UPnP Universal Plug and Play.

URL Uniform Resource Locator.

VLSI Very Large Scale Integration.

WAN Wide Area Network.

w.h.p With high probability.

WINS Windows Internet Naming Service.

WWW World Wide Web.

XOR eXclusive OR.

About the Author

The author holds a degree of Diplom-Ingenieur (Dipl.-Ing.) from the Technische Universität Berlin (TU Berlin), where he was studying computer engineering (Technische Informatik) at the department of Elektrotechnik & Informatik from October 1996 to February 2002. Since March 2002, he is a research assistant and a doctoral student at the DAI-Labor, a research laboratory of the Technische Universität Berlin.

Within the scope of this dissertation, the author has published the following articles to conferences and a Journal.

1. Giscard Wepiwé, Plamen L. Simeonov. A Concentric Multi-ring Overlay for Highly Reliable P2P Networks. *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA'05).* July 2005. Cambridge - MA - USA, Pages 83-90. ISBN: 0-7695-2326-9.

2. Giscard Wepiwé, Sahin Albayrak. A Churn-resistant Strategy for a Highly Reliable P2P System. *In Embedded and Ubiquitous Computing - IFIP EUC 2005 Workshops, Nagasaki, Japan, December 6-9, 2005, Proceedings, volume 3823 of Lecture Notes in Computer Science.* Pages 756-765. Springer Verlag.

3. Giscard Wepiwé, Plamen L. Simeonov. HiPeer: A Highly Reliable P2P System. *IEICE Transactions on Information and Systems.* Vol.E89A, No.2. Pages 570-580. February 2006.