Iowa State University
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2005

Protocols for sharing computing resources and dealing with nodes' selfishness in peer-to-peer networks

Rohit Gupta
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation

Gupta, Rohit, "Protocols for sharing computing resources and dealing with nodes' selfishness in peer-to-peer networks " (2005). *Retrospective Theses and Dissertations*. 1862.
https://lib.dr.iastate.edu/rtd/1862

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

Protocols for sharing computing resources and dealing with nodes' selfishness in peer-to-peer networks

by

Rohit Gupta

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Arun K. Somani, Major Professor
Akhilesh Tyagi
Tirthapura Srikanta
Thomas E. Daniels
Johnny Wong

Iowa State University

Ames, Iowa

2005

Copyright © Rohit Gupta, 2005. All rights reserved.

UMI Number: 3184617

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3184617

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Rohit Gupta

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

DEDICATION

To my grandparents, parents and my wife.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In the last decade Internet computing has been revolutionized, to a large extent due to applications such as file sharing developed around the peer-to-peer computing paradigm. Peer-to-peer (P2P) networks are flexible distributed systems that allow nodes (called peers) to act as both clients and servers to access and provide services to each other. In order to fully utilize the commercial potential of P2P paradigm and to make it more attractive to Internet users, the existing services of P2P networks need to be enhanced. Besides data sharing, P2P networks can provide distributed processing capability, whereby multiple peers can form a virtual private processing network by contributing their individual resources, such as idle (otherwise wasted) computing power, disk space, etc. Moreover, most of the current research in P2P systems is based on a cooperative network model. It is generally assumed that although there can be some malicious nodes in a system, most of the nodes are trustworthy and follow the protocols as implemented by the network designer. For example, almost all the proposed P2P lookup protocols assume that peers sincerely follow the routing protocol and forward messages for other peers in the network. Furthermore, resources, such as data and bandwidth, are assumed to be freely available. Such altruistic behavior (i.e., cooperative routing and free data uploads) of the service providers may not hold as P2P networks gain prominence and are deployed over large public networks, such as Internet.

Future P2P networks are likely to have no centralized administrative entity (owing to regulatory or scalability concerns) that control the nodes in the system. Therefore, it is important for network designers to take into account the independence and selfishness of P2P users to make the future systems more reliable and robust. Generally speaking, P2P network protocols need to be designed taking into account the fact that peers would behave selfishly to maximize their own interests or utility. The resources will generally be owned by different users or organizations that will not necessarily volunteer

to make them freely available, even when they are not being used. To provide the incentives for large-scale resource sharing, resources need to be buyable and sellable, with the possibility of contracts that (at least in theory) can be enforced.

A set of protocols are needed that range from ones that are very light-weight for small "micro-value" transactions, to those that are more heavy-weight where enforceability (perhaps at a later time) is more important than efficiency. This will support the creation of a market economy of network-accessible resources, including those that are specifically part of the network such as link bandwidth, and node processing and memory capacities. These resources would be purchased on demand, in very short periods of time, and for very short (and larger when necessary) periods of time. Users will rely more on buying the power in the network, only when and for how long they need it, rather than on relying solely on what is locally available. This will be more economical and offer the user more potential power. Moreover, market economic principles have been very successful in human societies for resource allocation, and therefore, we expect to derive similar benefits from computational resource economies built on top of P2P systems. as follows: To develop solutions for the above problems, we make the following contributions in this dissertation.

1. We provide a solution for enabling distributed computing by harnessing idle computing resources, such as CPU cycles, in P2P networks taking into account nodes' selfishness.

2. We develop a mechanism for pricing and trading resources such as data and routing bandwidth in P2P networks.

3. To increase users confidence to participate in the system, we develop a protocol for giving complete participation anonymity to the users of a P2P system. We also propose a reputation management framework for allowing users to evaluate the trustworthiness of each other before trading services/resources among themselves. The framework also implements a light-weight currency infrastructure that allows use of monetary incentive schemes for promoting cooperation among selfish users.

4. We use game theory concepts to model and investigate the behavior of users in P2P systems.

CHAPTER 1. INTRODUCTION

Unless you've been asleep at the wheel for the last nine months, you've heard of peer-to-peer (or P2P) computing. If you believe, as many do, that its use is limited to file sharing or that it is the most important development in the history of computing since the invention of the Internet, you can be forgiven – the hype generated by some of its more extreme proponents is to blame. In spite of the hype, P2P computing is important, and it's beginning to look like the paradigm with a large enough slice of mindshare to move a number of promising technologies from the wings into the limelight.

http://www-106.ibm.com/developerworks/java/library/j-p2p/

One of the latest buzzword in our computing world is P2P the short form of Peer-to-Peer Computing. When the old-time favorite B2B and B2C companies are folding shops and venture funds are moving away from internet businesses, new players continue to crowd the P2P space and there seems to be no shortage of venture capitals willing to bet on this promising technology.

http://www.theindianprogrammer.com/columns/pranab/p2p.htm

In the last decade Internet computing has been revolutionized, to a large extent due to applications, such as file sharing, developed around the peer-to-peer paradigm. The basic idea behind the P2P computing is simple - two computing devices (peers) share their information and resources together. No device is client or server; each one of them act as mini-servers. P2P model allows any group of persons connected to the network to directly communicate with each other to exchange any digital product, for example MP3 music, plain texts, games, videos, eBooks, etc.

P2P applications consist of a number of peers, each performing a specific role in the P2P network, in communication with each other. Typically, the number of peers is large and the number of different roles is small. Because of these two factors, P2P applications are characterized by massive parallelization in function. The best example is the Gnutella network, which consists of a large number of essentially identical peers. In developing P2P applications, the interesting problems lie in the interaction among peers and, to a lesser extent, in the peers themselves.

Peer-to-peer computing did not spring into existence in its current form. Rather, it is the child of a number of different parents. First and most important, P2P computing is the natural result of decentralizing trends in software engineering intersecting with available technology. From an engineering perspective, the trend over the last decade, driven by forces such as enterprise application integration, has clearly been away from monolithic systems and toward distributed systems. This trend was inhibited somewhat by the ease of managing centralized applications, but the growth of the Internet, followed by the rise in importance of business-to-business transactions, made full-scale distributed computing a business necessity. Intersecting this first trend is the second trend of the growth in the availability of powerful networked computers and inexpensive bandwidth. To be effective, P2P computing requires the availability of numerous, interconnected peers. These two trends combined to form the perfect playground for P2P applications research.

1.1 The Peer-to-Peer Paradigm

The computing world is witnessing a paradigm shift from mainframes to client-server models, and now to peer-to-peer computing, as highlighted in Figure 1.1

1970s and 1980s was the era of centralized computing, with IBM mainframe occupying over 70 percent of the world's computer business. Business transactions, activities and database retrieval, queries, and maintenance were all performed by the omnipresent IBM mainframes. In 1990s, we witnessed transition towards client-server computing. The main emphasis of client-server architecture is to allow large applications to be split into smaller tasks and to perform the tasks among host (server machine) and desktops (client machines) in the network. Client machines usually manage the front-end processes, such as GUIs (Graphical User Interfaces), dispatch requests to server programs, validate data entered by the user, etc. On the other hand, the server fulfill the client needs by performing the requested services. Typically a single server cater to the requirements of several (tens to thousands)

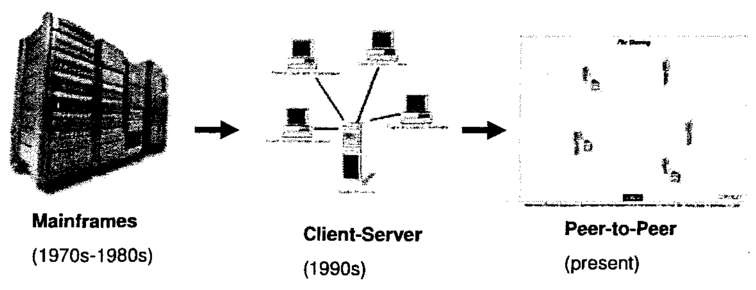


Figure 1.1 Evolution of computing systems.

client machines. This introduced a single point of failure and a severe strain on the resources of a single machine.

As the workstations and PCs became more powerful, and the Internet provided a powerful communication medium, it was realized that one can do away with a single server architecture and instead exploit the virtually unlimited computing capabilities of millions of machines connected to the Internet and local networks. This gradually led to the evolution of peer-to-peer networks. This paradigm shift is depicted in Figure 1.1.

Peer-to-peer (P2P) networks are flexible distributed systems that allow nodes (called peers) to act as both clients and servers to access and provide services to each other. P2P is a powerful emerging networking paradigm as it permits sharing of virtually unlimited data and computational resources in a completely distributed, fault-tolerant, scalable, and flexible manner. P2P systems can be used for various different purposes, such as distributed content management, distributed computing, etc.

In a typical content distribution P2P system, each peer has some resource to share. These resources can be files, documents, web pages, etc. Each resource is uniquely identified by a *key*. For example, a key can be a URL for a web page or the filename for a document. Given the key of a resource, or a set of keywords that describe a group of acceptable resources, the objective is to design a system that will locate resources in the system, and enable sharing with a minimum amount of communication, storage, and maintenance.

At the application level, P2P systems form a decentralized overlay network with its own routing mechanism. Due to its decentralized nature, the topology of the overlay network and its routing mechanism determine the system properties, such as performance, robustness, and scalability. Different P2P protocols organize peers with varying degrees of structure, from loosely structured systems (called unstructured P2P networks), such as Gnutella [1] and Freenet [2], to highly structured ones (called structured P2P networks), such as Chord [3], CAN [4], Pastry [5], Tapestry [6], etc. In unstructured P2P systems, data are stored anywhere in the system and are located by broadcasting queries to all peers within a specific distance. These methods are simple and robust to accommodate changes in the overlay network topology. However, the inefficiency of broadcasting affects their scalability. On the other hand, in structured P2P systems network topology and data placement are carefully designed in order to support efficient and scalable searching.

1.2 Evolution of Peer-to-Peer Systems

In some sense, peer-to-peer systems dates as far back as the origin of Internet. The Internet as originally conceived in the late 1960s was a peer-to-peer system. The goal of the original ARPANET was to share computing resources. The challenge for this effort was to integrate different kinds of existing networks as well as future technologies with one common network architecture that would allow every host to be an equal player. The first few hosts on the ARPANET–UCLA, SRI, UCSB, and the University of Utah–were independent computing sites. The ARPANET connected them together not in a master/slave or client/server relationship, but as peers.

The early Internet was also much more open and free than today's network. Firewalls were unknown until the late 1980s. Generally, any two machines on the Internet could send packets to each other. The Net was an instrument for cooperative research. The protocols and systems were obscure and specialized enough that security break-ins were rare and generally harmless. Some of the earlier successful applications based on the peer-to-peer technology are USENET [7] and FidoNet [8].

USENET, born back in 1979, is the distributed application that provide newsgroups. Its earliest incarnation was the work of two graduate students named Tom Truscott and Jim Ellis. At the time, nothing like the "on-demand" Internet we know today existed. Files were exchanged in batch over phone lines, often at night when long distance rates were lowest. Consequently, there was no effective way to centralize the function of the USENET. The natural result was an extremely decentralized, distributed application – a structure it retains to this day.

The other outstanding early peer-to-peer success is FidoNet. FidoNet, like USENET, is a decentralized, distributed application for exchanging messages. FidoNet was created in 1984 by Tom Jennings as a way to exchange messages between users of different BBS systems. Because it filled a need, it quickly grew and, like USENET, it remains in use today.

As can be seen peer-to-peer computing is not all that new. The term "P2P" is, of course, a new invention, but the basic peer-to-peer technology has been around at least as long as USENET and FidoNet – two very successful, completely decentralized networks of peers. However, P2P networks as we understand them today came into emergence with the introduction of Napster system.

In the following two subsections we describe various P2P systems categorizing them into two categories. We explain various designs that have been proposed for implementing P2P systems till date.

1.2.1 First Generation Systems

Napster [9] was the first publicly deployed peer-to-peer system. It used a central server to maintain an index of (*filename, IP address*) pairs, and kept track of the locations of all the music files in the system. When a user wanted a particular music file, it queried this server, and upon receiving the IP address(es) of the machine(s) that hosted the requested resource, it downloaded the file from that machine. The actual file transfer was independent of the central index server and just involved the two participating peers. The simplicity of Napster architecture partly contributed to its success and made it among the fastest growing software in Internet.

One potential argument against the simple architectural approach of Napster is the lack of scalability. Moreover, the setup of such systems may require considerable investment for high-performance machines (as index servers), high bandwidth, etc., which makes it unsuitable for implementing a peer-to-peer system. Furthermore, the approach is not fault-tolerant; the central server provides a single point of failure. This fact led to the ultimate demise of Napster as the legal authorities tracked, and eventually made the single entity that was responsible for the management of Napster to close down. Still, the tremendous success of Napster showed the power of P2P beyond doubt.

The fall of Napster saw the emergence of several other systems, such as Gnutella [1], Freenet [2], and KaZaA [10]. Like Napster, Gnutella decentralizes not only the file transfer, but also the initial resource discovery by flooding the resource request. Also, Gnutella is not restricted to only music files, one can share any type of files using the Gnutella protocol. Gnutella builds a network in which each machine is connected to a few other machines, called the neighbors, in the network. In other words, each machine knows the IP addresses of its neighbors and can communicate with them. If a user requests a file, the request is forwarded to one's neighbors; the neighbors in turn ask their neighbors, and so on. This process continues until the desired resource is found or the request has propagated the pre-specified number of hops away from the requesting user. The resource (if found) is then sent along the reverse lookup path so that it reaches the intended recipient. It is easy to realize that, unlike Napster, the Gnutella model is difficult to shut down because of its completely decentralized nature. Gnutella is also an open source under the terms of the GNU public license. The problem with Gnutella architecture is that of the search time. The search may take longer as the community grows. This is because it creates an enormous volume of traffic in the network for each request. Moreover, it is possible to have a failed request despite the fact that the resource resides in the network (since the search diameter is bounded by a time-to-live value).

KaZaA alleviates the problem of flooding traffic somewhat in practice by using *super-peers*. Super-peers are selected for their larger capacity and greater capabilities from among the set of peers. This approach essentially creates a hierarchical overlay network, where the top layer consists of the super-peers, and the bottom layer consists of the peers. All the resource requests are sent only to the super-peers, each of which maintains an index of the resources hosted by its own group. If a resource is not found in its own group, a super-peer communicates with other super-peers to locate the requested resource. Although it alleviate some of the problems of flooding, this approach is still inherently un-scalable. Flooding can be avoided, but with the additional cost of maintaining synchronized, distributed indices among the super-peers.

1.2.2 Second Generation Systems

Several recent systems like Chord [3], CAN [4], Pastry [5], and Tapestry [6], Koorde [11], Broose [12], P-Grid [13], Symphony [14], Viceroy [15], Yappers [16] use a distributed hash table (DHT) as the basic data structure for a peer-to-peer system. The main operation of a DHT is to retrieve the identity of a node that hosts a particular resource, starting from any other node in the network. DHTs allow location of resources using a directory-like interface that supports storing and fetching data indexed by keys. The underlying theme of all these systems is that they build an overlay network on top of the physical network, and embed the machines in the overlay network by hashing their identifiers. Resource keys are distributed, either randomly or by hashing, among the nodes to facilitate uniform load distribution. As both machines and resources are embedded in the overlay network using hashing, these systems are called distributed hash tables. Each node is responsible for the resources (i.e., either store the resource value or the address of some node that has the resource value) that hash to locations near itself.

The nodes are linked in the overlay network using a specific link distribution depending on the design. Resource location, using the overlay network, is done in these various systems by using different routing algorithms. Each node maintains some information about its neighbors, and routing is done greedily by forwarding messages to the neighbor closest to the target node. This inherent common structure leads to similar results for the performance of such networks; with n nodes in the network, most of these systems use $O(\log n)$ space at each node, and take $O(\log n)$ time for routing messages.

1.2.3 Current Research Efforts

Now we outline some of the major issues that the researchers in P2P community have primarily focussed on and addressed during the past few years. This gives the readers an insight into the trends in P2P computing research, and serves as a basis for us to provide the motivation for the research direction pursued in this dissertation.

Scalability: The Internet user community has grown to be so large that distributed systems need to cope with millions of users. In an ideal peer-to-peer system, the cost borne by each participant should not grow faster (in fact much less) than the size of the system.

Load balancing: The cost of maintaining the system should be uniformly shared between all the peers. Similarly, the system should be able to manage high data request volume, when a particular resource becomes extremely popular for a short period of time. For example, the popular web site CNN (http://www.cnn.com) saw record traffic, hitting nine million page views an hour during the terrorist attacks on September 11, 2001 in New York, USA, compared to an ordinary volume of eleven million page views a day [17].

Dynamic maintenance: The massive parallelism in peer-to-peer systems, due to high rate of machine arrivals and departures, present some very challenging issues that are trivially solved in a system with fixed membership. The system should be self-configuring, and machines and resources should be added and removed from the system without manual intervention.

Fault-tolerance: The overlay structure should be resilient to both machine and link failures in the system. Even if a part of the system has failed, the data available in the surviving machines should still be accessible, as long as it is located in the same connected component as the requesting peer. Further, the system should gracefully degrade in performance with increasing failures.

Self-stabilization: Not only should the system survive disruptions due to failures, but it should also heal automatically to restore ideal performance. The system should have a repair mechanism that detects local inconsistencies, such as machine failures or link outages, and triggers maintenance operations with minimal overhead in terms of network traffic.

Efficient searching: The primary goal of a peer-to-peer system is to locate resources efficiently, and hence support for searching using a variety of specifications is a very desirable property. Complex queries to locate resources, such as range queries, near matches to a key, and keyword matches should be supported by a rich query language.

Security: The system should be secure against attacks, such as a denial-of-service (DoS) attack, where some miscreant participants may flood the system, thereby preventing legitimate traffic. In some applications, it may also be desirable to maintain anonymity of the users, or provide resistance to censorship by preventing certain data items from being deleted from the system.

Topologically sensitive construction: Routing should be sensitive to network locality, such as distance travelled, or latency along transmission paths. Two possible approaches are - a) Proximity routing - machines are placed in the overlay network to exploit the underlying topology, and b) Proximity neighbor selection - the closest neighbor (as per the proximity metric) is chosen among the set of potential neighbors.

1.2.4 Future Requirements

Almost all the current (deployed or proposed) P2P systems are used primarily for *data/file sharing*. Although, it is widely acknowledged that other resources, like compute power can also be shared using the P2P paradigm, research in this regard is still underway. SETI@Home [18] comes close to sharing spare computing power (in the form of idle CPU cycles) of computers in a network. However, the model employed is still quite centralized. This is because SETI@home allows only a single server to make requests and use idle processing power of other computers in the network. The same capability is not available to all the participants of the SETI@Home network. Other research implementations such as CFS [19] and PAST [20] build large-scale distributed file systems on top of Chord [3] and Pastry [5], respectively. However, the needed processing is still performed by the clients after procuring the needed data.

In order to fully utilize the commercial potential of the P2P paradigm and to make them more attractive to Internet users, we believe that the existing services of P2P networks need to be enhanced. Besides data sharing, P2P networks can provide distributed processing capabilities, whereby multiple peers form a virtual private processing network by contributing their individual resources, such as idle (otherwise wasted) computing power.

Moreover, most of the current research in P2P systems is based on a cooperative network model. It is generally assumed that although there can be some rogue (malicious) nodes in a system, most of the nodes are trustworthy and follow the protocols as implemented by the network designer. For example, almost all the proposed P2P lookup protocols assume that peers sincerely follow the routing protocol and forward messages for other peers in the network. Future P2P networks are likely to have no centralized administrative entity (due to scalability concerns) that control the users in the system. Therefore, it is important for network designers to take into account the independence and selfishness of P2P users, in order to make the future systems reliable and robust. In fact, as pointed out in [23], "selfishness" is one of the main threats that can be expected to be faced in a deployed P2P system. Below is what the author of [23] says -

"*... Mojo Nation's experience shows that there are two kinds of attack that are likely to be encountered by any network that is deployed in a large scale on the Internet. The first attack is when a user alters his client in the attempt to gain more advantage for himself. Several different users have made such modifications to their Mojo Nation software and then helpfully contacted us to describe what they did. Other users have made modifications, but we are aware of those changes only indirectly through observations of anomalous behavior...*"

In some sense allowing users to act so as to maximize their own interests benefits the system as a whole; it allows for truly decentralized control and freedom for innovation - new users with new kinds of behavior and capabilities may enter the network. Generally speaking, P2P network protocols need to be designed taking into account the fact that peers would behave selfishly to maximize their own interests or utilities. The resulting protocols would be robust, fault-tolerant, and minimize free-riding.

Furthermore, resources, such as data and routing bandwidth, are assumed to be freely available. This assumption is inherent in the design of most of the current P2P networks, where one obtain services from others without having to pay any reward to the service provider. Such altruistic behavior, i.e., cooperative routing and free data uploads of the service providers, is not correct to assume as P2P networks gain prominence and are deployed over large public networks, such as the Internet. It has been pointed out that free-riding [21, 22] is one of the most significant problems being faced by today's P2P networks. It was found that 70 percent of the Gnutella users share no files and 50 percent of all responses are returned by the top one percent of sharing hosts [21]. Uncontrolled or too much free-riding leads to the degradation of system performance and adds vulnerability to the system. In order to avoid free-riding and achieve efficient allocation of resources (such as data, processing power, routing messages, etc.) made available by the P2P systems, we believe that the sharing of resources must be based on suitable economic models. The models employed should be scalable and have minimum (or no) requirement for any trusted and/or centralized entity.

Thus, although the issues described in Section 1.2.3 are critical for the feasibility and wide-spread deployment of the P2P systems, we believe that the capability for providing novel services (such as distributed computing), and taming the selfishness of individual users using market economic models would go a long way in fulfilling what the P2P paradigm promises to be.

1.3 Problem Statement and Organization of the Dissertation

Based on the issues identified above, the problems addressed in this dissertation consist of two primary components.

Problem 1: Provide a solution for enabling distributed computing by harnessing idle computing resources, such as CPU cycles, in P2P networks taking into account nodes' selfishness.

Problem 2: Develop a mechanism such that resources like data and routing bandwidth can be priced and traded in P2P networks.

1.3.1 Summary of the Chapters

Chapter 2 presents the system model used in this dissertation.

Chapter 3 presents CompuP2P, which is an architecture for distributed computing that facilitates trading of computing resources, taking into account nodes' selfishness, in dynamically created markets.

Chapter 4 proposes a unified incentive strategy that motivate intermediate nodes to route lookup requests and server nodes to share their data objects with others. Since, downloading a data item incurs a cost, the proposed strategy is an effective solution for dealing with the free-riding problem prevalent in P2P networks.

Chapter 5 proposes a reputation management framework for large-scale P2P systems. The framework assumes that every node behaves selfishly, and in addition there might be malicious nodes in the system. The proposed framework can also be used to implement a system of virtual currency by using reputation as a measure of nodes' wealth. This has tremendous potential as it obviates the need for an electronic payment infrastructure, and thus make the resulting system truly distributed and inexpensive.

Chapter 6 uses game theory to model the behavior of nodes in a P2P system and derives the Nash equilibrium of the underlying system. The derived Nash equilibrium addresses the problem of free-riding, since it shows that the best strategy for nodes is to probabilistically share their resources with others.

Chapter 7 presents a novel protocol for providing anonymity in P2P networks. The protocol is inherently anonymous, light-weight, and incentive-compatible. Incentive compatibility implies that the protocol takes into account the selfishness of users; as we would see the utilities of users are maximized by truthfully following the protocol steps. Moreover, unlike other schemes, the proposed protocol does not rely on any trusted centralized entity or require specialized encryptions to be performed by the users. Thus, the protocol incurs very low overhead on the system and is light-weight.

Chapter 8 summarizes and discusses some open issues for further research.

CHAPTER 2. SYSTEM MODEL

2.1 Overview

In this chapter we describe the system model used in this dissertation. However, if some additional assumptions are needed to the model to take into account the specific limitations of some of the proposed solutions, then those assumptions are described in the chapters where the respective solutions are presented.

Our system model is inspired by the principles of algorithm mechanism design and is analogous to the model presented by Nisan et. al. in [24]. Below we provide an introduction to the model developed in [24], which also provides a definition of algorithm mechanism design.

2.1.1 Model for Selfish Agents By Nisan et. al.

The view taken in the paper is that of a system's engineer that has certain technical goals for the global behavior of the network. The authors view the selfishness of the participants as an obstacle to such a goal that can be overcome by way of trade and payments. In economic terms, the desire is to develop a virtual *managed economy* of all network resources, but due to the selfishness of the participants one is forced to obtain it using the *invisible hand* of *free markets*. Therefore, the goal is to design the market rules as to ensure the desired global behavior.

The model presented allows studying these types of issues. The model relies on the rationality of the participants and is game-theoretic in nature. Specifically, it is based upon the theory of mechanism design [32]. The field of mechanism design aims to study how privately known preferences of many people can be aggregated towards a "social choice". One of the motivations to use mechanism design to solve computational problems is to deal with situations involving the differing goals of the participants.

The model is concerned with computing functions that depend on inputs that are distributed among

N different agents. A problem in this model has, in addition to the specifications of the function to be computed, a specification of the goals of each of the agents. The solution, termed a mechanism, includes, in addition to an algorithm computing the function, payments to be handed out to the agents. These payments are intended to motivate the agents to behave *correctly.* A mechanism solves a given problem by assuring that the required outcome occurs, when agents choose their strategies as to maximize their own selfish utilities. A mechanism needs thus to ensure that players' utilities (which it can influence by handing out payments) are compatible with the algorithm.

More formally, the mechanism design problem and its solution (called a mechanism) are defined as follows:

Definition 1. *A mechanism design problem is given by an output specification and by a set of agent's utilities. Specifically:*

- *Each of the N agents, say agent i has a private input value $t^i \in T^i$ (termed as the type of an agent).*
- *The output specification gives a set of outcomes o for every combination of agents' types.*
- *An agent attaches a valuation (v^i) to every outcome and the total utility of an agent (u^i) is the sum of v^i and the payment (p^i) that it receives from the mechanism because of that outcome.*

The model studies only optimization problems. In these problems the outcome specification is to optimize a given objective function. More formally, the outcome specification is given by a positive real-valued objective function $g(o, t)$ and a set of feasible outcomes F. The required output is the outcome $o \in F$ that minimizes g.

Definition 2. *A mechanism $m = (o, p)$ is composed of two elements: an outcome o for the given set of actions of the participants, and an N-tuple payments $p^1 \ldots p^N$. Specifically:*

- *The mechanism defines for each agent a set of strategies from which it can choose its action.*
- *The mechanism provides an outcome based on the set of actions of the participants.*
- *The mechanism provides a payment to the participants based on the outcome derived above.*

- *The mechanism is an implementation with dominant strategies, if for each possible set of dominant strategies (in a game-theoretic sense) of the participants, the outcome satisfies the specification.*

The simplest types of mechanisms are those in which the agents' strategies are to simply report their types. Further, if for a problem there is a mechanism designed to handle the dominant strategies of the participants, then one can also come up with a mechanism, which provides that truthfully revealing the input type is the best strategy for the participants.

2.1.2 Model Details

We assume a peer-to-peer network (either structured or unstructured) given by $G = (V, E)$. V is the set of nodes or peers and is given by $V = \{n_1, n_2, \ldots, n_N\}$. Here $N(= |V|)$ is the set size or total number of peers in the network. (We use the terms nodes, peers, and users interchangeably in this dissertation). The edges in set E are bi-directional and represent the neighbor relation between pair of nodes, i.e., we have $e_{ij} \in E$ if and only if nodes n_i and n_j are neighbors in the overlay network. $\mathcal{N}(n_i)$ represent the neighbors of a node, say n_i, and is given by $\mathcal{N}(n_i) = \{n_j | e_{ij} \in E, \forall j\}$.

The network is dynamic as peers join and leave at unpredictable times. We assume that each node has a unique public-private key pair and the public key of any node can securely be obtained by using its IP address.

Message communication is reliable, i.e., a message sent is received by the intended receiver in bounded time without any distortion. This assumption is made so as to simplify the description of the proposed protocols. If a network is not reliable, then message loss can be detected by setting suitable timeout values at the receivers. The sender is then assumed to have failed or left the network.

Nodes, are autonomous rational agents in a game-theoretic sense. By autonomous we mean that nodes are completely free to choose their actions. The goal of the rational agents in a game is to maximize their *utilities* (or *profits*) during each network transaction (or a game). The profit from a transaction is equal to the difference between the *reward* that a node earns and the *cost* that it incurs by participating in the transaction. Nodes participate in a transaction if there is a potential of making profit in future.

The *reward* can be anything that is deemed to have value, the possession of which adds to a node's utility. (The terms money, currency, and reward should be understood to mean the same thing). For simplicity, we assume that rewards are electronically processed and a secure payment mechanism among peers is in place [25, 26, 27, 28, 30, 29, 31]. The cost in the form of bandwidth, memory, etc., that a node x incurs by participating in a transaction is referred to as its *marginal cost* MC_x. The term "marginal" reflects the fact that this cost value is for a given transaction and represents the additional work that a node has to do for participating in the transaction. The cost incurred by a node increases in proportion to the amount of traffic it is currently handling, and any request offering less than its current marginal cost is not fulfilled.

A *transaction* is any event wherein some peer requests *service* from the network.

The definition of a *service* can include anything from sharing data, compute power to routing messages, etc. We say that a transaction t is an instance of some service, say $\mathcal{S}$. For example, for service $\mathcal{S}$ equal to *music file sharing*, the lookup process initiated by a peer to download a music file corresponds to a transaction. (The lookup process refers to the process of searching for and obtaining a desired resource).

There is a well-defined protocol, denoted by $P_{\mathcal{S}}$, for carrying out service $\mathcal{S}$. $P_{\mathcal{S}}$ is designed such that it satisfy the goals ($G_{\mathcal{S}}$) associated with the successful completion of service $\mathcal{S}$.

Each transaction is a game among the participating peers. The outcome of the game determines the increase in each peer's utility. The rules of the game are specified by $P_{\mathcal{S}}$. If it can be proven that any peer(s) did not follow the rules in $P_{\mathcal{S}}$, it (they) can be punished. We assume that there is a punishment mechanism in place, and nodes prefer not to get punished as compared to any other outcome of the game. In an enterprise computing environment there might be a central authority one can report to in order to identify and punish the cheating node. For large-scale open systems one can use reputation mechanisms to ensure that cheating nodes are identified and prevented from receiving any service in future.

Our approach: The way we address the problems - *Problem 1* and *Problem 2*, is by designing $P_{\mathcal{S}}$, such that each transaction not only satisfies $G_{\mathcal{S}}$, but also maximizes each node's expected utility at the same time.

Our approach is a distributed implementation of the mechanism described in the previous subsection. This is because our solutions require that it is in nodes' best interest to report their true marginal costs while participating in a transaction. Moreover, the collection of input values (i.e., marginal costs) and handing out of rewards to nodes is done in a distributed manner rather than by some centralized entity.

CHAPTER 3. "CompuP2P": AN ARCHITECTURE FOR DISTRIBUTED COMPUTING IN INTERNET-SCALE PEER-TO-PEER NETWORKS

3.1 Overview

The idea of a "network computing" system is gaining popularity over more traditional ones like high-performance multiprocessors and supercomputers. This is due to increased scalable cumulative power, more efficient (and economical) use of existing resources, and more effective system management. Ultimately, large economies of scale are achievable by the ability to arbitrarily deliver and redirect the shared power of the network to any user, in contrast to today's systems where power is essentially statically distributed and preallocated, and not shared.

CompuP2P is an architecture for enabling distributed computing in Internet-scale peer-to-peer networks. It provide resources, such as processing power, disk space, etc., of millions of PCs and workstations that have Internet connectivity and are under-utilized most of the time to user applications that might require them. For example, such a system can perform compute intensive tasks on behalf of clients, such as wireless devices (e.g. PDAs) with limited battery and processing power. Applications, like scientific simulations and data mining, requiring large processing power, can tremendously benefit from potentially unlimited availability of compute power provided by CompuP2P. Likewise, database applications, requiring huge storage, can harness the disk capacity of virtually millions of machines connected to the Internet. We believe that while the Internet currently cannot hope to serve as a totally general-purpose efficient parallel computer, it can still serve as an excellent platform with unlimited computational resources for solving a wide variety of computational problems. We sketch some of these application in Section 3.6.2.

At present, P2P networks, such as KaZaA [10], Gnutella [1], etc., are used primarily for data sharing. Although, it is widely acknowledged that other resources, like compute power, can also be shared using a P2P paradigm, research in this regard is still underway. SETI@home [18] comes close to sharing computing power (in the form of idle CPU cycles) of computers connected to the Internet. However, the model employed is still quite centralized. This is because SETI@home allows only a single server to make requests and use idle processing power of other computers in the network. The same capability is not available to all the participants of the SETI@home network. On the other hand, CompuP2P enable all the users to harness almost unlimited computing resources of the entire network.

The prospect of harnessing a significant fraction of network resources to execute a distributed application is very appealing. These resources can be purchased on demand, in very short periods of time, and for very short (and larger when necessary) periods of time. Resources will generally be owned by different users that will not necessarily volunteer to make them freely available, even when they are not being used. To provide the incentives for large-scale resource sharing, resources need to be buyable and sellable, with the possibility of contracts that (at least in theory) can be enforced.

To meet the above requirements, CompuP2P use light-weight protocols for building and operating dynamic computing resource markets, where sellers and buyers can come together to negotiate transfer (usage) of resources from seller to buyer nodes. The lookup of such markets and the availability of resources are robust even in the face of several nodes entering or leaving the network at the same time. CompuP2P use ideas from game theory [52] and microeconomics [53] to devise incentive-based schemes for motivating peers to share their idle computing resources with each other. The trading and pricing of resources is done in a completely distributed manner without requiring any trusted centralized authority to oversee the transactions.

For concreteness, in this chapter we use compute power as the resource under consideration, however, the mechanisms for market creation and resource pricing are equally applicable to any other kind of resource, such as disk space, etc.

3.2 Related Research

In this section we discuss some well-known projects that aim to harness the idle processing capacity in distributed systems. Almost all of these projects are based on grid computing and employ several centralized and/or trusted components for their task distribution and resource management. We also compare them to our proposed CompuP2P architecture.

3.2.1 Condor [33]

Work on Condor was started in 1988 by the Computer Sciences Department at the University of Wisconsin-Madison with the aim of developing a general-purpose framework that would allow the use of idle CPU cycles for research purposes. Condor is designed to support the execution of independent tasks following the SPMD (single process multiple data) model. It provides a flexible platform-independent framework for distributing *jobs* (tasks) over a pool of machines (peers) by providing a basic job queuing mechanism, scheduling policies, priority schemas and resource monitoring and management. It is built on the principle of distributing batch jobs around a loosely coupled cluster of computer to enable a high-throughput computing (HTC) system.

Users submit their serial or parallel tasks to Condor in form of jobs. The Condor matchmaker places them into a queue, chooses when and where to run them based on job needs, machine capabilities and usage policies. Condor monitors the progress of jobs and informs the user upon completion of their jobs. Condor uses a variety of different concepts to ensure fast and safe execution of jobs.

To protect the host, all jobs are executed in a restrictive sandbox that prevents/intercepts invoking any system calls. Only remote system calls are permitted since they will be executed on the host of the job's owner. In addition to this Condor supports strong authentication, encryption, integrity assurance, as well as authorization.

Condor harvests the otherwise wasted CPU power of desktops, workstations, servers and clusters. Condor is designed primarily for SPMD, but also supports MPMD (multiple process multiple data). Condor also has a large number of inter-job communication libraries, e.g. MPI [35].

Task management is centralized and ensures that jobs are executed in an efficient and secure manner based on the specified requirements of provider and consumer. However, a consumer has little control over the location and manner in which its job is executed.

3.2.2 Entropia [34]

Entropia is a commercial product, and is sold as part of Entropia's DCGrid enterprise solution (www.entropia.com). It was designed at the beginning of 2000, and completed by the middle of 2001. Entropia has been deployed as an enterprise desktop grid [37] at more then a dozen commercial sites, and used for more than 50 applications. Since the majority of desktops are Windows x86 machines, Entropia focuses purely on providing a Windows x86-based solution, supporting three generations of Windows operating systems NT, 2000, and XP.

The Entropia system aggregates the raw desktop resources into a single logical resource. This logical resource is reliable, secure, and predictable despite the fact that the underlying raw resources are unreliable (machines may be turned off or rebooted), insecure (untrusted users may have electronic or physical access to machines), and unpredictable (machines may be heavily used by the desktop user at any time). This logical resource provides high performance for applications through parallelism while always respecting the desktop user and his or her use of the desktop machine. Furthermore, the logical resources are managed from a single administrative console. Addition or removal of desktop machines from the Entropia system is easily achieved, providing a simple mechanism to scale the system as the organization grows or as the need for computational cycles grows.

The Entropia system architecture is composed of three separate layers. At the bottom is the Physical Node Management layer that provides basic communication and naming, security, resource management, and application control. On top of this layer is the Resource Scheduling layer that provides resource matching, scheduling, and fault-tolerance. Users can interact directly with the Resource Scheduling layer through the available APIs or alternatively, users can access the system through the Job Management layer that provides management facilities for handling large number of computations and files.

To support the execution of a large number of applications, and to support the execution in a secure manner, Entropia consist of three main components - Desktop Control, Sandbox, Application Security. Entropia uses a light-weight interface which mediates between the application processes and the operating system. This mediation is used to control the application behavior, and hook in other parts of the security system. It also uses an external process on the desktop machine, which is called the Desktop Controller. To monitor and control all of the application processes and threads running on the machine, Entropia uses the mediation to implement a sandbox which manages the execution of the application so that it can do no harm to the PC and so that it leaves the desktop PC in the same state as before the system was run. The application security is provided through a combination of a device driver, Information Technology support and automated file encryption. Creating a system to run native binaries under a virtual machine provides a key advantage, which is the ease of application integration to enable an application to run on the system.

Unlike Condor and Entropia, CompuP2P is completely decentralized, in the sense that there is no centralized entity that monitors system state and assign (sub)tasks accordingly to different machines. CompuP2P use microeconomic principles and game-theoretic ideas to govern trading and allocation of compute power to tasks.

3.2.3 Spawn [36]

At a very high level of description, Spawn is organized as a market economy composed of interacting buyers and sellers. The commodities in this economy are computer processing resources; specifically slices of CPU time on various types of computer workstations in a distributed computational environment.

Buyers are users who wish to purchase time in order to perform some computation. Sellers are users who wish to sell unused, otherwise wasted processing time on their computer workstations. A buyer can be a scientist who wants to run a large, concurrent Monte-Carlo simulation. A typical seller is a user who is not actively using his personal workstation. A seller executes an auction process to manage the sale of his workstation's processing resources, and a buyer executes an application that bids for time on nearby auctions and manages its use of computer processing resources. In the Spawn economy, monetary funds encapsulate resource rights, and price equates the supply and demand of processing resources.

Spawn is meant to be used in relatively small single-site distributed networks organized as a cluster. On the other hand, CompuP2P is designed to run in large geographically dispersed networks. In Spawn all the CPU cycle auctions are homogeneous and a bidder is indifferent to which node provides the necessary computing power for processing its tasks, whereas CompuP2P create different compute power markets and sellers lookup a specific market that can fulfill its processing requirements. Since, nodes in a P2P network can be far apart from each other, buyers in CompuP2P are also sensitive to the physical location of the sellers and try to minimize the time it takes to finish the entire computation. The central concern in Spawn is to ensure fair allocation of idle resources among concurrent applications and not locating and trading available compute power in a large Internet scale network.

3.2.4 POPCORN [38]

POPCORN enables any programmer on the Internet with a simple virtual parallel computer. This virtual machine is implemented by utilizing all processors on the Internet that care to participate at any given moment. In order to motivate this participation, a market based payment mechanism for CPU-time is employed. The system is implemented in Java and relies on its ubiquitous *applet* mechanism for enabling wide scale safe participation of remote processors.

The POPCORN programming paradigm, used by the buyer program, achieves parallelism by concurrently spawning off many sub-computations, termed *computelets.* The POPCORN system automatically sends these computelets to a market (chosen by the user), which then forwards them to connected CPU-time sellers who execute them and return the results. The matching of buyers and sellers in the market is dynamic, is done according to economic mechanisms, and results in a payment of the buyer to the seller.

The system is intended for coarse-grained parallelism. The efficiency is mostly determined by the ratio between the computation time of computelets to the communication effort needed to send them and handle the overhead. To achieve high efficiency, computelets should be relatively heavy in terms of computation time.

POPCORN provides an infrastructure for globally distributed computation over the whole Internet and uses a market-based mechanism to trade CPU cycles. However, POPCORN uses a trusted, centralized market that serves as a matchmaker between the seller and buyer nodes. On the other hand, markets in CompuP2P are distributed and dynamically created. Moreover, the market owners themselves are assumed to be selfish, which is a reasonable assumption in large Internet-scale systems with no pre-trusted entities.

3.2.5 SETI@Home [18]

SETI (the Search for Extraterrestrial Intelligence) is a collection of research projects aimed at discovering alien civilizations using radio telescopes. Since the analysis of the extensive radio telescope data (about 35GB per day) requires significant computing resources, a P2P approach for distributed computing was chosen. SETI@Home engages Internet users around the world in the effort of the distributed signal analysis.

SETI@Home server divides the data into chunks (work-unit) designed for an average desktop computer. Participating peers contact the server and download a chunk of data. After downloading the peer starts processing the data in its idle time, e.g. when the screen-saver is active. The result of the analysis is sent back to the central server and a new cycle of requesting data, processing data and reporting results begins.

The tasks in SETI@Home are independent and can be executed without the need of any connection. Network connectivity is only needed for receiving data and sending results. The peer data - including the number of work units completed, time of last connection, and team membership - are reported on web-sites allowing users to compete for the biggest CPU contributions.

SETI@Home uses a check-pointing mechanism to recover from inadvertent host or network failures. SETI@Home also injects "test signals" intentionally into the system to confirm that the hardware and software is working properly. The "suspicious" responses to work unit or the lack of reported results is recorded and used in evaluating the level of trust assigned to the peer.

SETI@Home is limited to SPMD problems and offers no communication support (except for requesting data and sending results). Hiding the details of the communication protocols, i.e., requiring that users install SETI@Home software prior to joining the network and avoiding any code migration provides basic security. SETI@Home uses redundancy and tests data to ensure correct processing and flags suspicious peers. Due to the large number of freely available computing resources no efforts for optimizing the execution of tasks is carried out.

In SETI@Home only one central server can allocate tasks to others, whereas the goal of CompuP2P is to enable all the peers to purchase computing power and distribute their workload onto other machines in the network. Thus, in this regard CompuP2P is a more flexible and powerful distributed processing paradigm.

The projects described above (including some other, such as [39, 40]), with the exception of SETI@Home and POPCORN, are based on grid computing, rather than true peer-to-peer computing. Nodes in these grid-based systems are under direct control of some form of centralized controllers. These centralized controllers have complete system information and schedule tasks onto multiple nodes after matching client requests with the resource capabilities of existing nodes. Moreover, selfishness of nodes is not an issue in these systems. On the other hand, although SETI@Home does not impose any direct control over the network nodes that agree to process sub-tasks, still the underlying architecture is quite centralized as only one node (SETI@Home server) can request others to perform tasks on its behalf. Also, POPCORN relies on a trusted centralized markets to enable matching and transfer of processing tasks from the client to server machines.

3.3 Model Assumptions

The network model uses Chord [3] for addressing and nodes' connectivity. We provide a brief description of the Chord protocol in Section 3.4. Although CompuP2P uses Chord as the underlying protocol, its architecture is generalized enough, such that with little modifications it can also be employed in other structured P2P networks such as CAN [4].

Typically in a network there are peers (called *computing nodes* or *sellers*) that may have idle computing resources available for executing tasks on behalf of other peers (called *clients* or *buyers*). We assume that nodes executing tasks get suitably compensated by the clients. Moreover, nodes behave selfishly and the only way for nodes to maximize their payoffs is by selling their idle computing resources to others that may require them. To simplify our discussion here, we do not explicitly consider compensating nodes for their data, however, in the next chapter we propose a distributed pricing strategy such that data can also be traded among nodes in a P2P network.

Now we discuss how *Problem 1* mentioned in Chapter 1 is dealt with by the CompuP2P architecture.

3.3.1 Problem Formulation

Service type (S): Utilizing idle computing resources in P2P networks.

Problem Definition: Provide a solution for harnessing idle computing resources, such as trading raw CPU power, in P2P networks taking into account nodes' selfishness.

Service Goals (G_S): We identify the following objectives for the service type defined above.

1. Nodes must be motivated to share their idle computing capacities with others.
2. A client (with sufficient funds) should always be able to locate a computing node and send its task(s) for remote execution, as long as the compute power requirement of the task(s) is smaller or equal in value to the existing idle capacity of some computing node(s) in the network.
3. Truth-telling, with regards to the marginal cost of providing compute power, should be the best strategy for the computing nodes.
4. Among all the available computing nodes, the one with the lowest marginal cost should be selected.
5. Payoffs received by nodes should be greater than their marginal cost of providing service.
6. A client should not be charged an arbitrarily high price for its task execution.

In the following section, we define protocol P_S for service S. P_S include the mechanisms for creation of computing resources markets and pricing strategies for trading of computing resources in those markets.

3.4 Chord Overview

Chord supports just one operation: given a key, it determines the node responsible for that key. Each Chord node has a unique m-bit identifier (ID), obtained by hashing the node's IP address. Chord views the IDs as occupying a circular identifier space. Keys are also mapped into this ID space, by

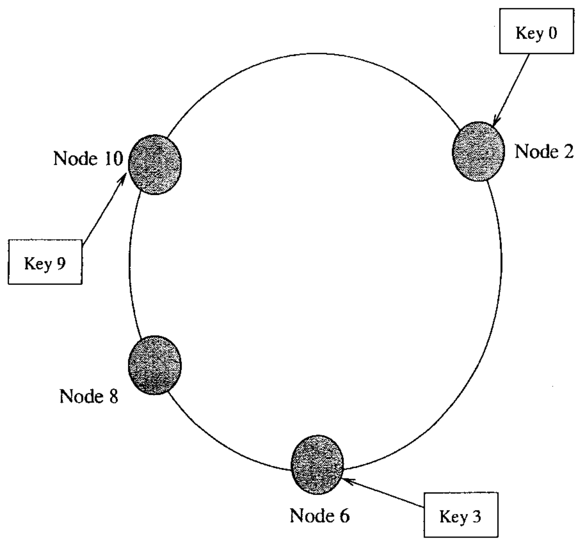


Figure 3.1 Mapping of keys to nodes in Chord.

hashing them to m-bit key IDs. We will use the term "key" to refer to both the original key and its image under the hash function, as its meaning will be clear from the context. Similarly, the term "node" refers to both the node and its identifier under the hash function.

Chord defines the node responsible for a key to be the *successor* of that key's ID. The *successor* of an ID *j* is the node with the smallest ID that is greater than or equal to *j* (with wrap-around), much as in consistent hashing [41]. A pictorial representation of a Chord network is given in Figure 3.1.

Consistent hashing lets nodes enter and leave the network with minimal movement of keys. To maintain correct successor mappings when a node n_i joins the network, certain keys previously assigned to n_i's successor become assigned to n_i. When node n_i leaves the network, all of n_i's assigned keys are reassigned to its successor. No other changes in the assignment of keys to nodes need occur. Also, consistent hashing does a good job of load balancing keys onto nodes. Intuitively, this follows since the use of an appropriate hash function means that node and key identifiers can be treated as independent, uniformly distributed random points on the circle. Consistent hashing is straightforward to implement, with constant time lookups, if all the nodes have an up-to-date list of all other nodes. However, such a system does not scale. Chord provides a scalable, distributed version of consistent hashing.

A Chord node uses two data structures to perform lookups - a successor list and a finger table.

Only the successor list is required for correctness, so Chord is careful to maintain its accuracy. The finger table accelerates lookups, but does not need to be accurate, so Chord is less aggressive about maintaining it. The following discussion first describes how to perform correct (but slow) lookups with the successor list, and then describes an accelerate technique using a concept called finger table.

Every Chord node maintains a list of the identities and IP addresses of its r immediate successors on the Chord ring. The fact that every node knows its own successor means that a node can always process a lookup correctly - if the desired key is between the node and its successor, the latter node is the key's successor; otherwise the lookup can be forwarded to the successor, which moves the lookup strictly closer to its destination.

A new node n learns of its successors when it first joins the Chord ring, by asking an existing node to perform a lookup for n's successor; n then asks that successor for its successor list. The r entries in the list provide fault-tolerance - if a node's immediate successor does not respond, the node can substitute the second entry in its successor list. All r successors would have to simultaneously fail in order to disrupt the Chord ring, an event that can be made very improbable with modest values of r. An implementation should use a fixed r, chosen to be $2\log_2 N$ for the foreseeable maximum number of nodes N.

The main complexity involved with successor lists is in notifying an existing node when a new node should be its successor. The stabilization procedure described in [3] does this in a way that guarantees to preserve the connectivity of the Chord ring's successor pointers.

Lookups performed only with successor lists require an average of $N/2$ message exchanges. To reduce the number of messages required to $O(\log N)$, each node maintains a finger table with m entries. The i^{th} entry in the table at node n contains the identity of the first node that succeeds n by at least a distance of 2^{i-1} on the ID circle. Thus every node knows the identities of nodes at power-of-two intervals on the ID circle from its own position. A new node initializes its finger table by querying an existing node. Existing nodes whose finger table or successor list entries should refer to the new node find out about it by periodic lookups.

The following two theorems, given in [3], show that neither the success nor the performance of Chord lookups is likely to be affected even by massive simultaneous failures. Both theorems assume that the successor list has length $r = O(\log N)$. A Chord ring is *stable* if every node's successor list is correct.

Theorem 1. *In a network that is initially stable, if every node then fails with probability 1/2, then with high probability the search for a successor returns the closest living successor to the query key.*

Theorem 2. *In a network that is initially stable, if every node then fails with probability 1/2, then the expected time to find a successor is* $O(\log N)$.

3.5 Markets for Trading of Computing Resources

We now explain how nodes in CompuP2P, possibly across different administrative domains, can share their idle computing resources, specifically compute power. Each node based on its current and past load estimates the average number of CPU cycles that would remain idle in future.[1] Suppose a node determines that it has C cycles/sec available for the next T time units (where T is some large enough time period) that it can provide or make available to others for processing.[2] These available CPU cycles can be time shared across multiple tasks, as long as the sum of the requirements of all the tasks do not exceed C. For example, if C is equal to 10^5 cycles/sec, then a node can execute a task that needs at most 10^5 cycles/sec, or if there is no such single task, the processing power may be time-shared among multiple tasks given that the total requirements of the tasks do not exceed 10^5 cycles/sec. It must be noted that the same value of number of CPU cycles/sec might represent different amounts of compute power for different nodes. This might happen if nodes have different hardware and/or software configurations. We use the unit of cycles/sec to represent normalized equivalent amounts of compute power at different nodes in a heterogeneous system.

Once the amount of idle computing resources has been estimated, the next step is to determine how to sell them. Moreover, buyers needing extra computing resources should be able to locate the right sellers and purchase the resources from them. The related and equally important issue is how the sellers should price their resources in order to maximize their profits. In the next subsection, we first describe

[1]For example, by using information from Unix commands, such as "top" and "uptime".

[2]In case some other resource, say disk space, is under consideration then we would use another appropriate unit, like G gigabytes for T time units.

techniques for dynamically creating and locating markets, such that no single node is overburdened with the task of maintaining and running the markets.

3.5.1 Constructing Markets for Buying and Selling Computing Resources

Since different nodes have different amounts of compute power to sell and purchase, it is necessary to create suitable markets to permit buyers and sellers to come together and trade the amount of compute power they require. For a buyer to sequentially search the entire Chord ring for the best available deal is a very time consuming and expensive operation. Also, selecting one node, say the successor of Chord ID zero, where all the transactions for all the available compute power in the network take place is not a good idea either. This is because relying on one node can lead to extreme scalability, fault-tolerance, and security problems.

For efficient creation and lookup of compute power markets, we propose two schemes that attempt to uniformly distribute the location of and responsibility for maintaining those markets across the network. Both the schemes use Chord for market assignment and lookup, however, they differ from each other in the overhead involved and the manner in which nodes are selected for running markets for various commodities. The term *commodity* as used here represents a range of idle CPU cycles/sec values. Each market deals in only one type of commodity (i.e., homogeneous markets). A single physical node may be responsible, i.e., be a market owner (*MO*), for more than one market.

Figure 3.2 depicts how nodes with different values of idle compute power C join different markets. Although, for simplicity of discussion, we have used C as a discrete value, in actual practice it refers to a well-defined range of values within which a node's idle processing capacity can lie. Thus, nodes with different but close enough idle processing capacities trade in the same market.

We describe below two schemes for the creation of compute power markets.

3.5.1.1 Single Overlay Scheme

In this scheme, the value C computed by a seller acts as the Chord ID for locating the corresponding compute power market. The successor node of Chord ID C is assigned the responsibility for maintaining the market for that particular idle compute power. It is possible that several compute power values

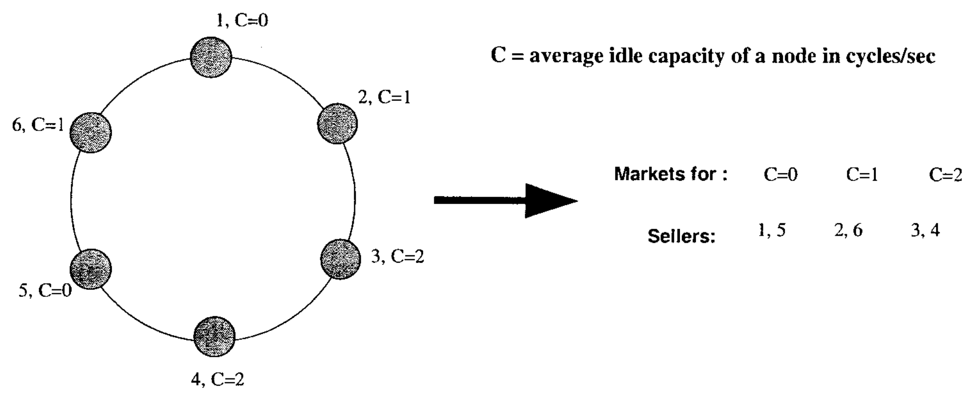

1, C=0
2, C=1
3, C=2
4, C=2
5, C=0
6, C=1
C = average idle capacity of a node in cycles/sec
Markets for : C=0 C=1 C=2
Sellers: 1, 5 2, 6 3, 4

Figure 3.2 Creation of markets for CPU cycles in CompuP2P.

map to a single node and then that node is responsible for running different markets, all dealing in different commodities.

This scheme is very simple to implement and involves almost no additional overhead to create markets. Compute power markets are searched using the normal Chord lookup protocol. In other words, if a node needs to purchase x cycles/sec, it simply looks up for the market maintained by the successor of Chord ID x. The drawback of this scheme is that if the idle compute power values in the network happen to be in a very narrow range, then most of the markets map to only a very few distinct physical nodes. Those nodes can then become the bottleneck and degrade the system performance. Moreover, search for a suitable market by a buyer might potentially require several attempts. In each attempt the amount of compute power searched for is successively increased, until a desired seller with adequate capacity is discovered.

3.5.1.2 Processor Overlay Scheme

In order to more uniformly distribute the responsibility for running the compute power markets and to bound the search time for an appropriate seller, an additional overlay can be maintained that keeps information about available idle compute power at different sellers in the network. All *MO*s, which are responsible for various commodities, constitute this Chord-based overlay network. The total ID space of this new overlay is equal to the maximum amount of compute power that may possibly be available on any single node and is upper-bounded by $2^c - 1$, where c is a constant and represents the number of bits used to represent the maximum value of idle CPU cycles/sec. We assume that the value of c is large enough to represent the idle processing power of even a very large computer system.

The process of selecting a *MO* for a commodity is illustrated in Figure 3.3. A node on determining its value of C applies a hash function to C to find the corresponding Chord ID (= $hash(C)$, a value between 0 and $2^m - 1$). The successor node of $hash(C)$ is then the *MO* for the market trading in commodity C. The various *MO*s defined in this manner then together form another overlay network, called the *processor overlay*, which has ID space from *0* to $2^c - 1$. The ID of a *MO* in this new overlay network is simply the value C whose hash value was mapped to it in the initial Chord network. Stated otherwise, the ID of a *MO* in the processor overlay network, called CPU Market ID (*CMID*), is the number of CPU cycles/sec that are being sold in its market.

It must be noted that in the above description, it is possible that a single node in the initial overlay network is the *MO* for several different markets, causing it to have multiple *CMID*s assigned to it in the processor overlay network. Each *CMID* value is represented by a different node in the processor overlay, as shown in Figure 3.3. *MO*s (i.e., nodes comprising the processor overlay) periodically send out a broadcast message identifying themselves to nodes in the network. Each node needs only store information about a single *MO* (or at most a few for fault-tolerance) to be able to perform lookups in the processor overlay.

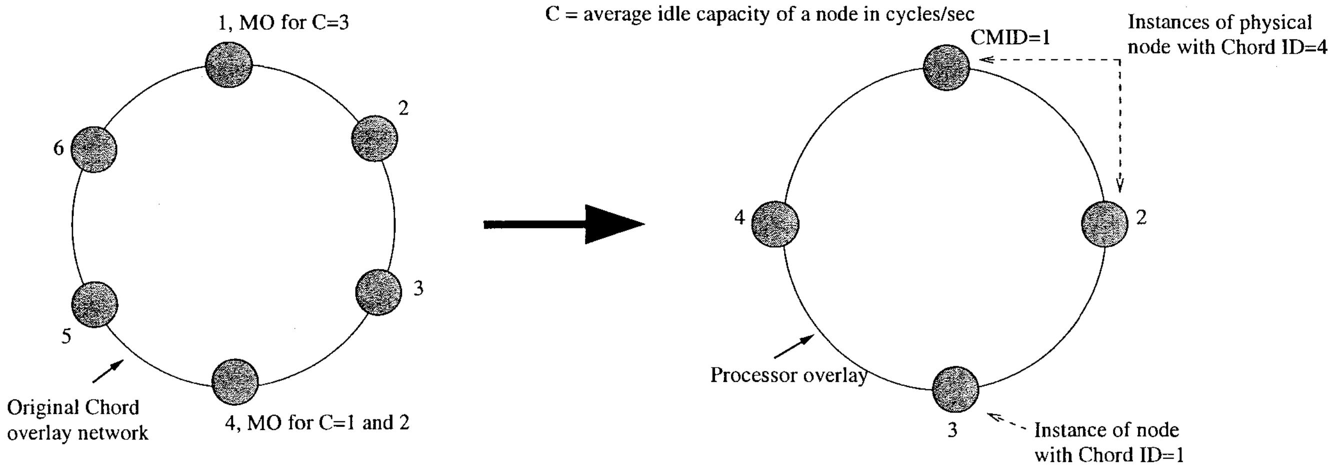

1, MO for C=3
2
3
4, MO for C=1 and 2
5
6
Original Chord overlay network
C = average idle capacity of a node in cycles/sec
CMID=1
Instances of physical node with Chord ID=4
2
3
4
Processor overlay
Instance of node with Chord ID=1

Figure 3.3 *Processor overlay* schema using the CPU capacity values given in Fig. 3.2.

The lookup in processor overlay, require $\frac{1}{2}(\log M)$ hops on average, where M is the number of different markets. Moreover, *MO*s store additional $O(\log M)$ routing information to support the Chord protocol for market lookups.

The search for the compute power in the processor overlay scheme is performed based on the number of CPU cycles/sec (which acts as the lookup key) that a client requires for processing. The client first contacts any of the known *MO*s and forwards the lookup request to it. The selected *MO* searches for an appropriate market for the desired compute power in the processor overlay network. The lookup process finally returns the IP address of the *MO* that runs the market for that compute power, or the nearest higher compute power value available in the network. For example, if only two compute power markets (with commodity values b and c) exist in the network, and a client desires a (where $a < b < c$), then the above mechanism returns the market for b instead of c. The *MO* is then contacted to obtain information about the sellers listed in the market.

Nodes have incentive to become *MOs*, since they make profit by charging *listing price* (*LP*) from sellers (and/or buyers) that benefit from the services provided by a market. We describe below two pricing schemes that can be used by a *MO*.

- A *MO* can charge the same fixed price to all the sellers that are listed in the market. This is a simple strategy, however, since there is no central authority to govern the listing price, the *MO* can charge arbitrarily high prices to the sellers and/or may price discriminate among them. Moreover, this scheme also does not take into account the dynamics of a particular market. It seems unfair that sellers should pay the same listing price, when in fact they earn different profits depending on the market they are in and the existing competition. We refer to this scheme as *fixed listing pricing*.

- A *MO* can charge (to the buyers or sellers or both) on the basis of the market characteristics, say some percentage of the selling price. This scheme appears to be fair to both the sellers as well as the *MO*, since a seller is not required to make a payment till it is able to sell its compute power, and the *MO* also potentially gets a higher payoff depending on the dynamics of the market. Although appealing, this simple scheme in fact can be difficult to implement in a distributed setting when the participants (buyers, sellers, and market owners) are selfish. We refer to this scheme as *variable listing pricing*.

3.5.2 Pricing for Computing Resources

Pricing is non-trivial when there are either multiple at par sellers from a buyer's point of view or when a buyer is trying to minimize its cost of processing (again assuming multiple sellers). Utilizing the model that a transaction involving the trading of compute power can be modelled as a one-shot game and using the results from game theory and microeconomics (the classical Prisoner's dilemma problem [52] and Bertrand oligopoly [53], respectively), we can see that long-term collusion among compute power sellers (and *MO*) is unlikely to occur. In one-shot Prisoner's dilemma game, non-cooperation is the only unique Nash equilibrium strategy for the players. In fact, the model of Bertrand oligopoly suggests that sellers (irrespective of their number) would not be able to charge more than their marginal costs for selling their resources (see [52] for a game-theoretic derivation of this result). In Bertrand oligopoloy sellers strategy is to set "prices" (as opposed to "outputs" in Cournot oligopoly), and is thus more reasonable to assume in the context of CompuP2P. In CompuP2P all the sellers in a market sell the same amount of a computing resource. As a consequence, sellers, irrespective of how many there are in a market, in CompuP2P set prices equal to their marginal costs only.

One-shot model of a compute power transaction is reasonable to assume, since once a seller sells its compute power, it de-lists itself from the market and perhaps move to another market for selling its remaining compute power, if available. Moreover, in a dynamic system, where nodes continually join and leave the network, it is difficult to keep track of nodes that do not fulfill their collusion agreements. Thus, nodes are not likely to be penalized based on their past behavior.

3.5.2.1 Providing Incentives to Sellers

Since the best pricing strategy for sellers is to charge equal to their marginal costs, it results in zero profits for them. Therefore, sellers would not be motivated to sell their computing resources unless some other incentive mechanisms are devised for them. Below we describe two such strategies depending on whether fixed or variable listing pricing is used to compensate a *MO*.

- **Strategy For Fixed Listing Pricing.** If fixed listing pricing is possible, then a *MO* has no incentive to cheat and thus we can use the technique employed in Vickrey auction [24, 54]. A seller when it joins a market provides its marginal cost information to the *MO*. A buyer, looking to minimize its cost, selects the seller with the least marginal cost, but the amount it has to pay to the seller is equal to the second lowest marginal cost value listed in the market. This selection scheme is called *reverse Vickrey auction.*

The above strategy provides non-zero profit to the selected seller and ensure that sellers state their correct marginal costs to the *MO* (see [54] for the truth-eliciting property of Vickrey auction). The strategy is also inherently secure because even if sellers learn about the posted marginal costs, they cannot take undue advantage of that information to post a lower marginal cost than their actual values. To understand this, consider the following simple example.

Example: Suppose seller *A* has the marginal cost (MC_A) of 5 and the lowest marginal cost among all the sellers different from *A* ($= {MC_A}^{-1}$) is 4. If *A* hides its true *MC* and posts it as 3 in order to get selected, its actual payoff would be (${MC_A}^{-1} - MC_A$) or 4-5 = -1, i.e., it would suffer a loss of -1. Thus, it can be seen that the only rational strategy for a seller is to post its correct *MC*. In this incentive scheme, a seller selected for processing makes a profit of $(MC^{-1} - MC)$.

- **Strategy For Variable Listing Pricing.** If variable listing pricing is being used, the above scheme based on Vickrey auction cannot be employed. This is because Vickrey auction is designed to be used by non-selfish auctioneers (here *MO* is the auctioneer), whose goals are to maximize system efficiency as opposed to personal gains. Whereas, in variable listing pricing, a *MO* has incentive to behave selfishly to maximize its profits. For the case of fixed listing pricing this selfishness was not a problem, since the payoff that a *MO* received was fixed. But if the payoff that a *MO* receives is dependent on a transaction outcome, then it has incentive to cheat. To understand how a *MO* may cheat consider the following example.

  Example: Let us say, a *MO* receives 10 percent of a transaction value from the sellers. Suppose there are three sellers, *A, B*, and *C* currently listed in the market. The marginal costs of *A, B*, and *C* are 100, 200, and 300, respectively. If a buyer now makes a request for the lowest cost supplier then the *MO* has incentive to report *C* as the lowest cost supplier, instead of *A*. This is because by doing so the *MO* earns a profit of 30 (=300*10/100) instead of 10 (=100*10/100).

Even if Vickrey auction is used, the *MO* has incentive to report 200 and 300, instead of 100 and 200 as the lowest and second lowest cost values, respectively, to the buyer.

In order to deal with the selfish *MO* problem, we propose a *max-min payoff* strategy. This strategy makes the payoff to a seller and *MO* complementary to each other, i.e., if the seller receives a high payoff than the *MO* receives a low payoff, and vice versa. We define the following simple model for this strategy. Let there be S sellers in a market, represented by $1, 2, \ldots, S$, such that $MC_i < MC_{i+1}$ for all $1 \leq i \leq S-1$. The sellers are not aware of each other (and of the buyers) and only know their own marginal costs, which they truthfully report to the MO. Buyers are also completely unaware about the sellers that are listed in the market and rely on the *MO* to give them information about the lowest cost supplier.

The payoffs to the *MO* and the selected seller by the buyer under max-min payoff strategy (based on the marginal cost values that a buyer receive from the *MO*) are as follows.

$$\begin{aligned} \text{Payoff}_{\text{MO}} &= (MC'_S - MC'_1)/(MC'_S)^2 + \delta \\ \text{Payoff}_{\text{seller}} &= MC'_1 + 1 \end{aligned} \tag{3.1}$$

MC'_1 and MC'_S in the above equation refer to the marginal cost values of the lowest and highest cost supplier, respectively, as reported by the *MO* to the buyer. Note that a *MO* can manipulate the reported values if doing so increases its payoff. Also, δ is a fixed payoff that a *MO* receives from a buyer irrespective of the marginal costs of the sellers. Therefore, the *max-min payoff* strategy can be considered to implement a hybrid listing pricing that has features of both fixed as well as variable listing pricing.

The above payoff values guarantee that the total cost to the buyer is bounded, and the best strategy for the *MO* is to return the lowest cost supplier only. We formalize this in the form of the following lemma.

Lemma 1. *Assuming one-shot model of compute power transactions, the payoff strategy in Equation 3.1 guarantees the following.*

a) The lowest cost supplier is always selected.

b) The payoff received by the selected seller covers its marginal cost of providing the service.

c) The total cost to the buyer is bounded.

d) The payoff to the MO is variable depending on the dynamics of a market, specifically, it depends on the marginal costs of the sellers listed in the market.

Proof. a) The *MO* can increase its payoff by reporting a low value for the lowest listed marginal cost, i.e., minimizing MC_1' as much as possible. However, MC_1' cannot be decreased below MC_1, the true lowest marginal cost, since otherwise the seller (here seller *1*) gets a payoff of $MC_1' + 1 (\leq MC_1)$. Since a seller does not provide its service unless its payoff is greater than its marginal cost, the best strategy for the *MO* is to set $MC_1' = MC_1$ and return the lowest cost supplier for processing.

b) This is implied from Equation 3.1 where we see that the payoff received by the seller is one more than its marginal cost.

c) From Equation 3.1, the payoff to the *MO* is maximized for $MC_S' = 2 * MC_1$ (after setting $\frac{\partial \mathrm{Payoff_{MO}}}{\partial \mathrm{MC_S}} = 0$), giving it a payoff of $1/(4 * MC_1)$. Note that in the given network model, it is difficult for a buyer to verify the marginal cost values it receives from the *MO*. Thus, the total cost to the buyer is bounded, and is equal to $1/(4 * MC_1) + \delta + MC_1 + 1$.

d) It follows from the description of the payoff values given by Equation 3.1.

□

In the above we assume that a *MO* serve the buyers in the order in which it receive requests from them. Moreover, once a seller has been selected for processing, it de-lists itself from the market, and joins some other market if it has sufficient compute power remaining.

From Lemma 1 it can be seen that the max-min payoff strategy satisfy service goals 4, 5, and 7, as defined in Section 3.3.1. Moreover, the use of reverse Vickrey auction for fixed listing pricing, and max-min payoff strategy for variable listing pricing, enforces that truth-telling with regards to the marginal cost of providing compute power, is the best strategy for the computing nodes (service goal

3). Also, since payoffs received by nodes are more than their marginal costs of providing services, they are motivated to share resources with others (service goal 1). Furthermore, service goal 2 is satisfied because resource markets are constructed in such a manner that a client always finds a suitable seller if one exists in the network.

3.6 Prototype Implementation of CompuP2P

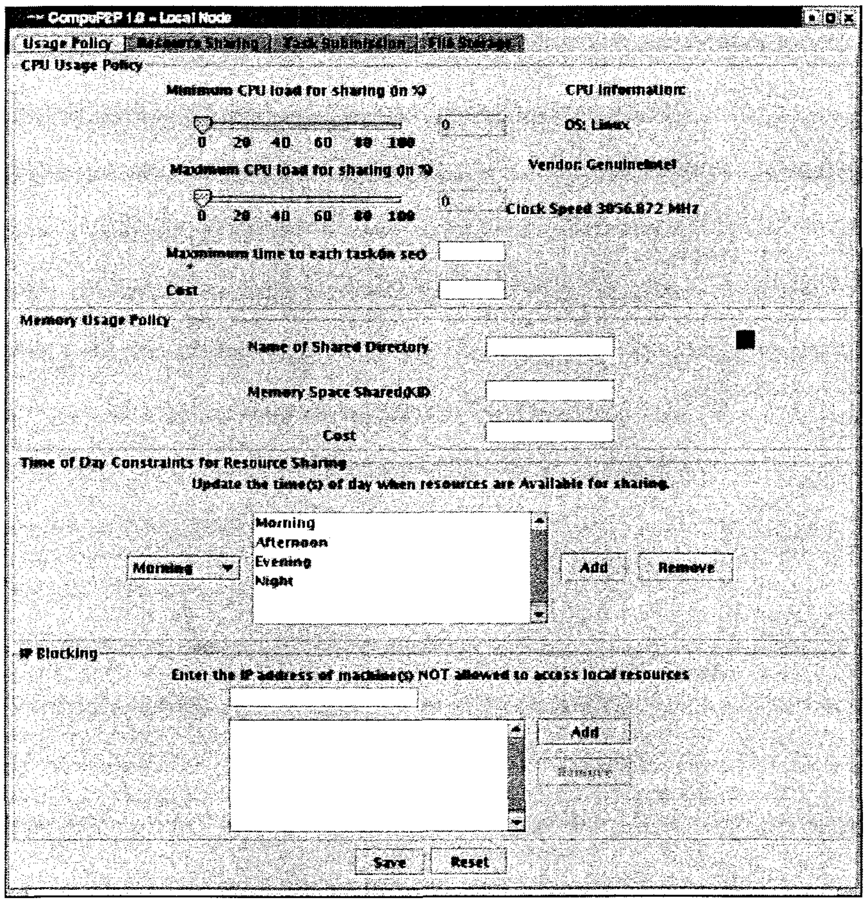
We have implemented a Java-based prototype of the proposed CompuP2P architecture for the sharing of compute power, and have deployed it in our lab for running compute intensive simulations.[3] Java owing to its platform independence and write-once run-anywhere feature enables easy migration of tasks from one node to another in a heterogeneous system.

Screen-snapshots of the implemented CompuP2P prototype, as it appears to a user, are shown in Figures 3.4 and 3.5. The first tab, "Usage Policy", allows a user to specify the usage constraints on the local shared resources. For example, a user can specify the CPU load levels (in percentage) beyond which the node is not allowed to share its compute power. To prevent a task from running forever, the user can also impose the maximum allowable run time on tasks received for execution. The user can also specify specific times of the day compute power can be shared. For example, one can specify that compute power can be shared only during night time when the machine is mostly unutilized. Moreover, for sharing storage space, the user can specify the total allocated shared space, along with the directory name where the received files are to be stored. Furthermore, the user can limit access to the machine by specifying the IP addresses of nodes that are not permitted to utilize the shared resources.

The second tab, "Resource Sharing", is divided into two components. In the first component the user specifies whether compute power and disk storage are shareable or not (usage policies described above are consulted before the resources are actually made shareable). The second component lets the user advertise files, which can be downloaded by others in the network.

The "File Storage" tab allows a user to back up its local data on multiple remote machines in the network.

[3]The work involving GUI snapshots development using Java NetBeans, CPU load calculations, and XML task file parsing is done by Varun Sekhri and is from [55].



CompuP2P 1.0 - Local Node

Usage Policy | Resource Sharing | Task Submission | File Storage

CPU Usage Policy

Minimum CPU load for sharing (in %): 0

Maximum CPU load for sharing (in %): 0

Maximum time to each task(in sec):

Cost:

CPU Information:

OS: Linux

Vendor: GenuineIntel

Clock Speed 3056.872 MHz

Memory Usage Policy

Name of Shared Directory:

Memory Space Shared(KB):

Cost:

Time of Day Constraints for Resource Sharing

Update the time(s) of day when resources are Available for sharing.

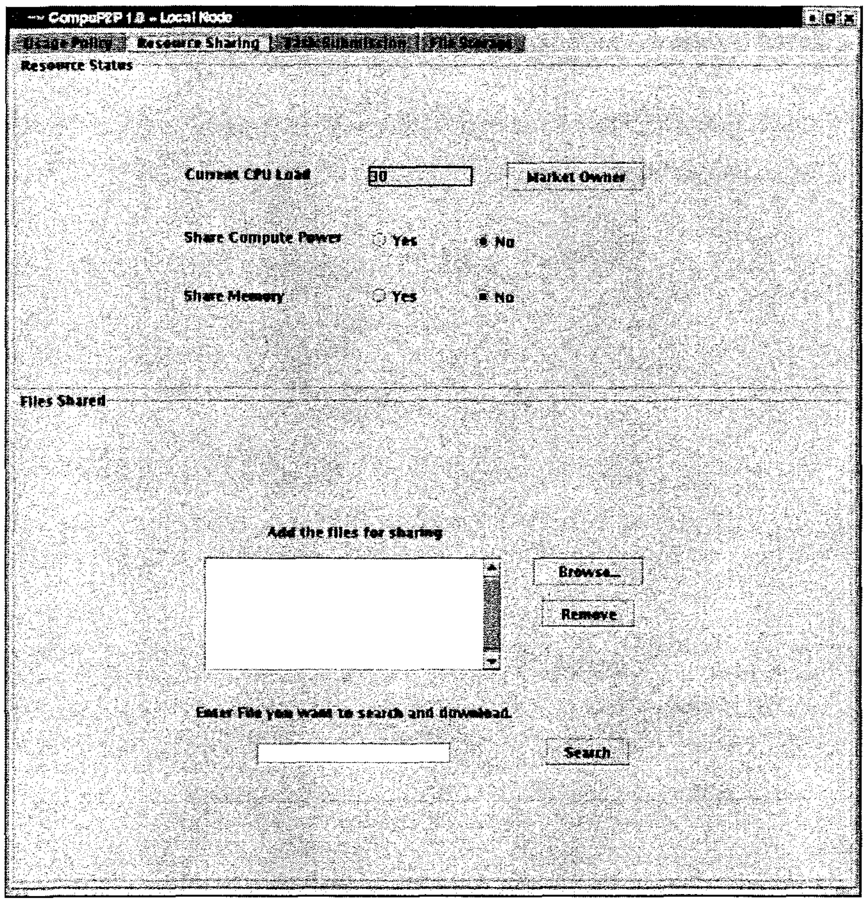
Morning

Morning
Afternoon
Evening
Night

IP Blocking

Enter the IP address of machine(s) NOT allowed to access local resources

Figure 3.4 Screen snapshot of Usage Policy tab.



Current CPU Load: 30

Share Compute Power: Yes [✓ No]

Share Memory: Yes [✓ No]

Figure 3.5 Screen snapshot of Resource Trading tab.

```
<? xml version="1.0" ?>
<TaskFile>

<SubTask>
  <CPUCycles>1000</CPUCycles>
  <Price>10</Price>
  <Checkpoint>true</Checkpoint>
  <CheckpointSize>50</CheckpointSize>
  <Code>job.exe</Code>
  <InputParameters>
    <Count>2</Count>
    <Param>Mesh.exp</Param>
    <Param>Mesh.dat</Param>
  </InputParameters>
  <OutputParameters>
    <Count>1</Count>
    <Param>result.out</Param>
  </OutputParameters>
  <Config>
    <OS>Linux</OS>
    <Processor>Intel Pentium 3</Processor>
    <Memory op=’gt’ unit=’MB’>256</Memory>
  </Config>
</SubTask>

</TaskFile>
```

Figure 3.6 XML-based task file

A user submits its task to the system in the form of a *task* file. The task file contains a description of various sub-tasks (a given task is assumed to be broken into several independent sub-tasks) that need to be solved. A sample task file is shown in Figure 3.6. For each sub-task, the following information is included.

- *Code ID* (or name) of the executable file for the sub-task. The executable file (if not locally available) can be downloaded either from a well-defined code server or can be searched for and downloaded just as other normal data using code ID (or name) as the key. To ensure security, the computing node executes the downloaded code in an appropriate sand-boxing environment. A sub-task is executed at a single node and thus define the level of granularity at which parallelism can be achieved.

- Names of input and output files to be used. If the input files are not available with a computing node, they can be searched for using the Chord lookup protocol.

- Estimated amount of compute power required.

- User's budget, i.e., the maximum amount of reward that a user can give in order to get the sub-task successfully executed.

- An indication whether the sub-task is to be periodically checkpointed or not, and the estimated size of checkpoint data.

- Platform specifications desired for selecting a seller.

The submitted task file is parsed and a thread is spawned for each sub-task. The thread created is responsible for looking up an appropriate seller node, negotiating the price for sub-task execution, and finally obtaining the results of computation and storing them in the output file(s) specified by the user. The task file is supplied to the service layer via the "Task Submission" tab.

A node first enters the network by contacting a bootstrap server running at a well-known IP address and port number. This bootstrap server is referred to as *AdminServer* in our implementation. AdminServer has information about all live nodes in the network, and returns the (IP address, port number) of a randomly selected existing node when contacted by a new node. The new node then uses this returned value to join the Chord network and update its routing table.

In our current implementation, we use AdminServer as a trusted bank that maintains an account for each node in the system. A node when it first enters the network is assigned some minimum currency that is credited to its account. Users' accounts are automatically debited (credited) by AdminServer whenever they buy (sell) compute power as per the pricing strategy outlined in Section 3.5.2. Buyers with insufficient balance are not permitted to use computing power of others in the network.

In addition to the GUI-based interface, one can use our *RemoteExecution* API for submitting a task file to the CompuP2P system. We have provided a TCP/IP socket interface for allowing the *RemoteExecution* API, which is in Java, to be usable by applications written in other languages. Applications supply the task file name over the socket connection, and are provided a notification (of success or failure) when all the sub-tasks defined in the task file are finished executing.

The system is clearly intended for very coarse-grained parallelism. The efficiency is mostly determined by the ratio between the computation time of sub-tasks to the communication effort needed to send them and handle the overhead. To achieve high efficiency, sub-tasks should be relatively heavy in terms of computation time.

In order to address the nodes' heterogeneity problem there are at least couple of possible alternatives. One alternative is to use the *SPECjvm98* benchmark [58]. A benchmark program can be selected based on the type of applications typically submitted by the users of the network. Benchmarks help to normalize the compute power values so that a given value is interpreted similarly by all the different nodes. These normalized values help to create homogeneous markets such that different sellers have equivalent compute power to offer, i.e., given a program all the sellers take approximately the same amount of time to execute it. To understand how this normalization is achieved consider the following example. Say, there are two nodes A and B that takes time T_A and T_B, respectively, to execute certain program P of the benchmark. If A has C_A and B has C_B available compute power, then the normalized idle compute power of A and B is given by C_A/T_A and C_B/T_B, respectively. These values are then used to determine the market they should join in order to sell their compute power. Another alternative that is currently used in CompuP2P is to use a *MO* as a *matchmaker.* A *MO* now stores the detailed platform description of all the sellers in the market. This description includes information such as OS type, OS version, processor configuration, etc. On receiving a request from a buyer, the *MO* selects the seller that not only has the lowest cost, but also meet the platform specifications as desired by a client for its sub-tasks.

3.6.1 Handling Failures of MOs and Listed Sellers

It is possible that nodes selected as *MO*s as well as sellers listed in those markets might fail. To account for such possibilities, we incorporate the following additional strategies in our prototype implementation.

1. Sellers periodically re-list themselves in a (new) market. This is done irrespective of whether the amount of a computing resource they are offering has changed or not. This periodic listing takes care of the following two problems that may arise in any dynamic system - a) *MO*s leaving the network, and b) new nodes joining the network that may replace some existing *MO*s as the new *MO*s for the respective markets.

2. Likewise, every *MO* periodically purges the information it maintain about the listed sellers. Thus, seller information is maintained as a soft-state information, and is never outdated for too long.

Additionally, a *MO* before returning information about the selected seller to the buyer checks whether the seller is still alive (i.e., is part of the network) or not.

3.6.2 CompuP2P Applications

We list here several of the applications of the CompuP2P system. These applications are ones that can utilize the idle computing power in P2P systems. A common characteristics of all these applications is that they all are loosely coupled, i.e., they can be broken into rather independent sub-tasks, each heavy in terms of processing power requirements, but relatively light in terms of communication requirements. Some of such application are *brute force search*, *code breaking*, *simulated annealing*, *task-parallel applications*, *parameter-sweep applications*, and *Monte-Carlo simulations*. All of these applications primarily involve generating many solutions in parallel and then using the solutions to come up with an answer for the initial problem.

We have found CompuP2P to be very useful for running large simulations. We used CompuP2P for running parallel simulations on multiple optical network topologies as generated by a user using ISTOS [59], which is an advanced tool for simulating fiber optic networks. Users in ISTOS can create multiple network topologies on which simulations are to be carried out. Earlier all simulations were sequentially executed on a single back-end server. (More information on ISTOS can be found in [59]). However, now we can exploit the CompuP2P architecture to distribute the task of simulating different network topologies to different nodes in the network that agree to share their idle computing power. We observed an almost perfect speedup in terms of the time required to finish simulation runs on all the submitted topologies, that is very close to the number of nodes that were used in parallel to run the simulations. This high speedup was possible mainly because of the low communication overhead - input to the computing nodes include a topology specification file and other parameters needed to start a simulation, and output include a file containing the simulation results (no communication among the computing nodes is required). (An executable file of the simulation code was pre-installed on the nodes).

Figure 3.7 shows the speedup achieved as a result of using the *RemoteExecution* API. In our experiments the number of computing nodes were 10, and the number of sub-tasks were successively increased from 1 to 10. It can be observed that CompuP2P provides substantial performance improvement and this is achieved by simply utilizing the idle capacity of machines in the network. The speedup would become close to 1 as the task computation time is increased. Ups and downs seen in Figure 3.7 for the parallel execution time are due to the heterogeneity of processors used in the experiment.

3.6.3 Fault-Tolerant Computing

It is possible that a computing node might not be able to finish the computation assigned to it, either because it leaves the network, it crashes, or the computation takes longer to complete than initially anticipated by a client. Under such circumstances, it may be expensive to restart the computation all over again. To handle such cases, it might be useful to periodically checkpoint the computing node's state, so that if required the failed computation can be migrated to another node in the network.

Unlike traditional checkpointing, which relies on dedicated checkpoint servers to store the processing state, we propose to use *server-less checkpointing* in which nodes that store the checkpoint data are determined on-the-fly. Similar to the techniques outlined in Section 3.5 for the sharing of compute power, we can construct markets for memory storage. The client based on its estimation of the amount of checkpoint data can reserve the required memory space on nodes, called *storage nodes.* The nodes performing computation are made aware of the storage nodes, to which they periodically send a

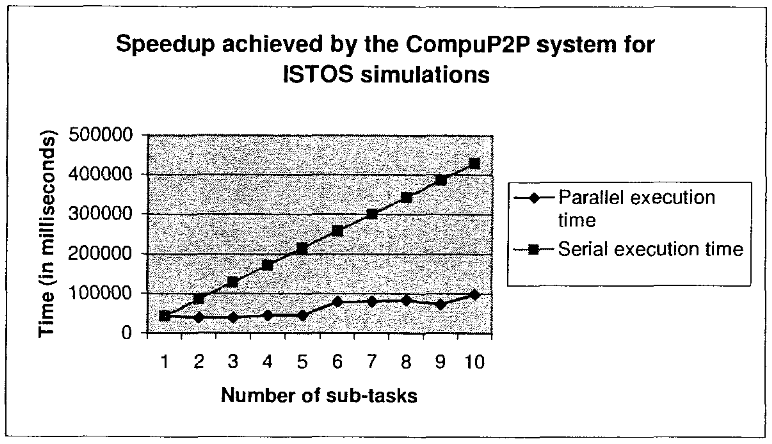


Figure 3.7 Figure showing the total time required to execute sub-tasks using CompuP2P, as opposed to the time required when all the computations are carried out on a single node.

checkpoint of their computations. Upon failure of a computing node, the stored checkpoint data can be used to re-start the computation at another suitable node in the network. We use the object serialization feature provided in Java, which enables a failed computation to be continued at a different node upon failure of an initially allocated processing node.

Our server-less checkpointing protocol is designed to take into account the failure of both the computing as well as the storage nodes. The fault-tolerance is achieved primarily by implementing two independent and parallel activities in CompuP2P, as illustrated in the figure below.

```
/* U is the user's service layer, C is the computing node, and S is the
storage node */

Activity 1:
U monitors C
if C fails
then
  Step 1: U finds another suitable computing node C'
  Step 2: Instructs C' to retrieve the last stored checkpoint data from S

Activity 2:
C monitors S
if S fails
then
  Step 1: C finds another suitable storage node S'
  Step 2: C notifies U, and gives the information about S'
```

Figure 3.8 Fault-tolerant checkpointing activities

Further, in practice errors in computation and/or communication of results can occur. Computation errors can occur due to faulty software/hardware at the computing node, or when a malicious node deliberately produces incorrect output. Such errors might be hard to detect and correct. To increase the reliability in the correctness of the end results the following alternatives can be used:

- Redundant computations (as also used in SETI@Home [18]) can be employed. Basically this scheme involves performing the same computation multiple times at different nodes and then selecting the result produced by the maximum number of the computing nodes.

- The tasks may be designed in a way that certain characteristics of the answer are known in advance to the client, but hard to deduce just from the task code. In these cases, an answer that has these characteristics may be assumed correct.

- Some tasks may return answers that are easily verified correct. For example, a task, which solves an equation using some complex method, may be easily verified by plugging the solution into the equation.

However, all the fault-tolerance features come at an increased cost to a user. The user's budget should be sufficient to cover the cost of reserving memory space to store the checkpoint data and/or compensate the redundant computing nodes for their processing.

3.6.4 CompuP2P Overhead

CompuP2P is a light-weight architecture and incurs minimal overhead on the system. It is built on top of Chord, which is a scalable, efficient, and robust protocol [3]. In this section we examine in detail the additional overhead incurred by the sellers, buyers, and *MO*s by the CompuP2P architecture. The overhead is in the form of either message communication or state maintained by each of these entities.

- Message communication overhead incurred by both buyers and sellers to locate a market is $O(\log N)$ in case of single overlay scheme, and $O(\log M)$ (where M is the number of different markets) for processor overlay scheme. Once a buyer has selected a seller for service further communication between them takes place using a direct TCP/IP connection, bypassing Chord routing. Message communication overhead incurred by *MO*s is almost negligible (apart from direct TCP/IP connections with buyers and sellers).

- In processor overlay scheme all nodes have to maintain additional (apart from the initial Chord-based overlay state) $O(\log M)$ routing information for maintaining the processor overlay, and in single overlay scheme no additional routing information is required.

  A buyer (seller) maintains the IP address of a seller (buyer). Moreover, information maintained by *MO*s is minimal. This information size is given by $n * a * s$, where n is the number of sellers in a market, a is the number of different attributes of a seller, and s is the space required to store a value of each attribute. To see how much the value of product $n * a * s$ evaluates to, let us consider a *MO* that stores information about 10,000 sellers. The *MO* might store several attributes pertaining to a seller. These attributes include information regarding a seller's IP address, marginal cost, OS type, OS version, processor configuration, etc. Suppose that there are 10 attributes value for each seller, and each attribute require 4 bytes of memory space. Then, the total information maintained by the *MO* is equal to, $10,000 * 10 * 4 \approx 400$KB. Thus, even for a large-sized market, its state information (< 0.5MB) can easily reside in any modern PC's RAM, which are typically 512MB. Furthermore, since the entire market information easily fits into a *MO*'s main memory, lookups to select an appropriate (based on a client's request) seller are also very fast.

3.7 Summary

In this chapter we described mechanisms for creation of computing resources markets and pricing strategies in those markets. The mechanisms proposed are completely decentralized, robust, and take into account nodes' selfishness. Specifically, the mechanisms described in Section 3.5 for creation of computing resources markets and pricing of computing resources, satisfy all the service goals that were identified in Section 3.3.1.

CompuP2P enables Internet-scale distributed computing and is significantly different from other large-scale distributed computing projects which have been implemented in the arena of grid or public resource sharing computing. CompuP2P can be used for building large Internet computing infrastructures, and can potentially reduce the need for expensive processing or storage servers in an enterprise, for example. Users of CompuP2P can harness almost unlimited processing capacity of the entire network in a completely distributed manner without relying on any centralized administrative authority.

CompuP2P relies on a monetary payment scheme to compensate nodes for their computing resources. While the use of a monetary scheme provides a clean economic model, implementing the associated electronic payment infrastructure can be very expensive. In order to overcome this problem, in Chapter 5, we propose a framework for implementing a system of virtual currency by using reputation as a measure of nodes' wealth.

3.7.1 Contrast with Grid Computing and Public Resource Computing

Internet computing along with grid computing and public resource computing share the goal of better utilizing existing computing resources. However, there are profound differences among the three paradigms, and it is unlikely that current grid middleware [62] or public resource sharing architectures [63] will be suitable for Internet computing.

Grid computing [37] involves organizationally-owned resources - supercomputers, clusters, and PCs owned by universities, research labs, and companies. These resources are centrally managed by IT professionals, are powered on most of the time, and are connected by full-time, high bandwidth network links. Malicious behavior, such as intentional falsification of results is handled outside the system, e.g. by carrying out a legal investigation.

Public resource computing [63] involves an asymmetric relationship between projects and participants. Projects are typically small academic research groups with limited computer resources, expertise, and manpower. Most participants are normal Internet users with PCs, workstations, etc., with low bandwidth connectivity to the Internet. The computers are frequently turned off or disconnected from the Internet. Participants contribute their resources either out of altruism or if they receive suitable "credit" for doing so. Projects have no control over participants, and cannot prevent malicious behavior.

In contrast, the Internet computing paradigm, implemented by CompuP2P, aims to create a single large heterogeneous pool of computing resources into which users can tap into to carry out their tasks. Here, users can include either enterprises, research groups, or even individual home PC owners. The system is typically large, may be millions of users, and network connectivity as in public resource computing is sporadic. There is no centralized entity that control the behavior of individual users, and thus users can be expected to behave selfishly (and even maliciously). Due to the large-scale, dynamism, openness, and heterogeneity of these systems, building a platform for Internet computing present several unique and interesting research challenges. Several of these issues, along with how they are addressed by CompuP2P, were discussed in this chapter.

CHAPTER 4. A LOOKUP STRATEGY FOR INCENTIVIZING SELFISH NODES TO SHARE DATA AND ROUTING BANDWIDTH

4.1 Overview

Almost all the current research in peer-to-peer (P2P) systems is based on a cooperative network model. It is generally assumed that although there can be rogue nodes in a system, most of the nodes are trustworthy and follow some specific protocol as suggested by the network designer. We believe that such assumptions do not always hold, and in large-scale open systems these issues have to be dealt with in order to make P2P systems reliable, robust, and to realize their true commercial potential. Moreover, it has been pointed out that free-riding, whereby only few altruistic nodes share their data, is one of the most significant problems being faced by today's P2P networks [21]. Solutions that exist to tackle this problem suffer from one of the following drawbacks - they are either too heavy-weight and expensive (for example, require trusted hardware), or depend on some trusted groups of nodes (or a trusted centralized entity) to police the network and keep the free-riders in check. Trust relationships are, however, difficult to establish in Internet-based P2P settings.

In this chapter, we describe a novel strategy for carrying out lookups and obtaining data in P2P networks with selfish nodes. Our approach does not require specialized hardware at each node, or prior trust relationships among nodes. Both the data provider and intermediate nodes that assist in routing of lookup messages are appropriately compensated so as to cover their cost of providing service. This is in contrast to traditional lookup schemes, which assume that data is freely available, and intermediate nodes cooperate and truthfully follow a given protocol in carrying out data lookups irrespective of whether they themselves are currently overloaded or not, for example. The proposed scheme provides an efficient and natural means to prevent the free-riding problem in P2P networks, and does not require prior trust relationships among nodes. Moreover, unlike other schemes it does not rely on any centralized entity or require specialized hardware at nodes. Therefore, the proposed scheme incurs low overhead and is highly robust.

The protocol proposed here is essentially an *incentive-driven lookup* protocol, which ensure that rewards received by intermediate nodes and data providers for routing and serving requests, respectively, are maximized by truthfully following the protocol steps. A distinguishing feature of the proposed lookup protocol is that it addresses the problem of incentivizing peers for sharing data and routing messages for others in a unified manner.

4.2 Related Research

The need for developing protocols for selfish agents (nodes) in P2P systems has often been stressed before also [24, 42, 43]. Broadly speaking, selfishness is the behavior exhibited by nodes when they access or use resources provided by others, without sharing their own for use by others. The resources in question can be data, CPU cycles, or bandwidth for forwarding messages. We explore some of the mechanisms that have been proposed in literature to address the problem of selfishness in P2P systems.

4.2.1 Mechanisms for Sharing Data

Proposals such as [44, 28, 45, 46] provide solutions to avoid the free-riding problem in P2P networks, but all these solutions are expensive and/or difficult to implement in a real-world setting. The basic approach in all these is to make sure that nodes indeed contribute to others before they themselves can obtain services from the network. Also, most of these solutions rely on self-less participation of different groups of nodes to monitor/police the activities of each node to ensure that everyone contributes to the system.

An interesting algorithm for achieving cooperation among network nodes is proposed in [47]. The algorithm does not require centralized or third party reputation systems, the monitoring of neighbor behavior, or the explicit programming of incentives, and operates in highly dynamic and noisy environments. The algorithm allow nodes to adapt selfishly and still maintain high levels of cooperation among them.

The basic algorithm assumes that peer nodes have the freedom to change behavior (i.e., the way they handle and dispatch requests to and from other nodes) and drop and make links to nodes they know about. Over time nodes engage in some activity and generate some measure of utility U (this might be number of files downloaded or jobs processed etc., depending upon the domain). Periodically, each node i compares its performance against another node j, randomly selected from the population. If $U_i < U_j$ node i drops all current links and copies all node j links and adds a link to j itself. Also, periodically, and with low probability, each node adapts its behavior and links in some randomized way using a kind of mutation operation. Mutation of the links involves removing all existing links and replacing them with a single link to a node randomly drawn from the network. When applied in a suitably large population, over time, the algorithm follows a kind of evolutionary process in which nodes with high utility tend to replace nodes with low utility (with nodes periodically changing behavior and moving in the network). However, this does not lead to the dominance of selfish behavior. This happens because although a selfish node may do well for a while it will tend to lose its exploited neighbors as they find other nodes that are members of more cooperative groupings and hence have higher utilities.

Although the above algorithm has several advantages, there are also significant limitations that can affect its performance. It is assumed that nodes would truthfully share their total utility values with each other. This is not a reasonable assumption since nodes are expected to behave selfishly. Moreover, it is not clear if a node can derive higher utility by deviating from the link removal/addition protocol described in the paper. Specifically, can a node derive higher utility by adding on links to new nodes without removing existing links, or by maintaining more than one link after the mutation operation.

Besides the above mechanisms for motivating nodes to cooperate and share their resources with others, several reputation management systems have been proposed as a way for incentivizing nodes to behave correctly and to minimize free-riding in the network. The goal of these systems is to identify (and isolate) uncooperative (and possibly malicious) nodes in the network. Several of such reputation management systems are discussed in Chapter 5.

4.2.2 Mechanisms for Sharing Bandwidth

To the best of our knowledge none of the existing solutions that deal with the problem of free-riding in P2P networks also address the more basic question of why nodes would route messages for others.

Since these nodes belong to end users without any centralized controlling authority, they may in order to conserve their bandwidth and other resources such as buffer space, memory etc., may drop messages received for forwarding. The mechanism proposed in [48] motivates nodes in P2P networks to share their routing bandwidth. [48] requires a node to route messages for others in order to obtain routing service in return. The authors in [48] develop a trust and security architecture for routing and node location service that uses a trust protocol, which describes how honest nodes should perform.

The problem of selfishness in routing has been encountered and addressed in the context of mobile ad-hoc networks (see [49, 50, 51]). Some of these proposals can also find application in P2P networks. Below we describe some of the recent research effort that promotes cooperation among selfish mobile ad-hoc network nodes.

In [49], Buttyan and Hubaux proposed a stimulation approach that is based on a virtual currency, called nuglets, which are used as payments for packet forwarding. Using nuglets, the authors proposed two payment models: the Packet Purse Model and the Packet Trade Model. In the Packet Purse Model, the sender of a packet pays by loading some nuglets in the packet before sending it. Intermediate nodes acquire some nuglets from the packet when they forward it. If the packet runs out of nuglets, then it is dropped. In the Packet Trade Model, the destination of a packet pays for the packet. To implement the Packet Trade Model, each intermediate node buys a packet from its previous node for some nuglets and sells it to the next node for more nuglets. In this way each intermediate node earns some nuglets and the total cost of forwarding the packet is covered by the destination.

Besides the above two models, Buttyan and Hubaux also proposed a scheme based on credit counter [50]. In it nodes pass each packet (generated as well as received for forwarding) to its security module. The security module maintains a counter, called *nuglet counter*, which is decreased when the node wants to send a packet as originator, and increased when the node forwards a packet. The value of the nuglet counter must remain positive, which means that if the node wants to send its own packets, then it must forward packets for the benefit of other nodes. The nuglet counter is protected from illegitimate manipulation by the tamper resistance of the security module. In this new scheme, each node keeps track of its remaining battery and its remaining credit. The authors simulated four rules (involving different degrees of cooperation) for a node to determine when to forward others' packets and when to send its own packets. The results showed that the most cooperative rule is actually optimal to achieve the goal of maximizing each nodes' throughput (and goodput).

Another approach is exemplified by the Sprite system [51], which provide incentive to mobile nodes to cooperate. When a node sends its own messages, the node looses its credit (or virtual money) to the network because other nodes incur a cost to forward the messages. On the other hand, when a node forwards others' messages, it gains credit. The system determines payments and charges from a game-theoretic perspective and is effective in motivating nodes to behave honestly, even when a collection of the selfish nodes collude. However, Sprite relies on a centralized trusted third-party to achieve its goals.

4.3 Model Assumptions and Limitations

We assume a P2P network model, wherein nodes act selfishly. Nodes process and forward lookup messages if there is a potential for making profit in future.

We assume that for each data there is a single server in the network, i.e., caching and replication of data does not take place. This can be enforced for example by using watermarking techniques [64]. We assume this so as to simplify the lookup mechanism and to be able to deterministically predict whether a given data is present in the network or not, and at which location. Caching and replication can cause ambiguities regarding the possible location of data, and therefore, for simplicity, we assume that there is a one-to-one mapping between a data item and its provider. Moreover, both caching and replication consume storage resources. Therefore, unless nodes are suitably compensated, they might not store data for others even if it is required by the protocol design.

Data indices are replicated at k different nodes, which are called the *terminal* nodes for that data. The lookup messages are first routed to terminal nodes from where they are sent directly to the server node (in one logical hop). Terminal nodes maintain a mapping (called index) from a data name or ID to the IP address of the server providing that data. For a data item, say D, its terminal nodes are denoted by $T_{D_i}\ \forall i \in \{1, \dots, k\}$. (Here D denotes the name or ID of some data item). The routing of a message from a client to a terminal node may go through other intermediate nodes. This list of intermediate nodes along with the terminal node is referred to as a *request chain*. For simplicity request chains comprising of different terminal nodes for the same data are assumed to be node disjoint (except at the client), as shown in Figure 4.1.

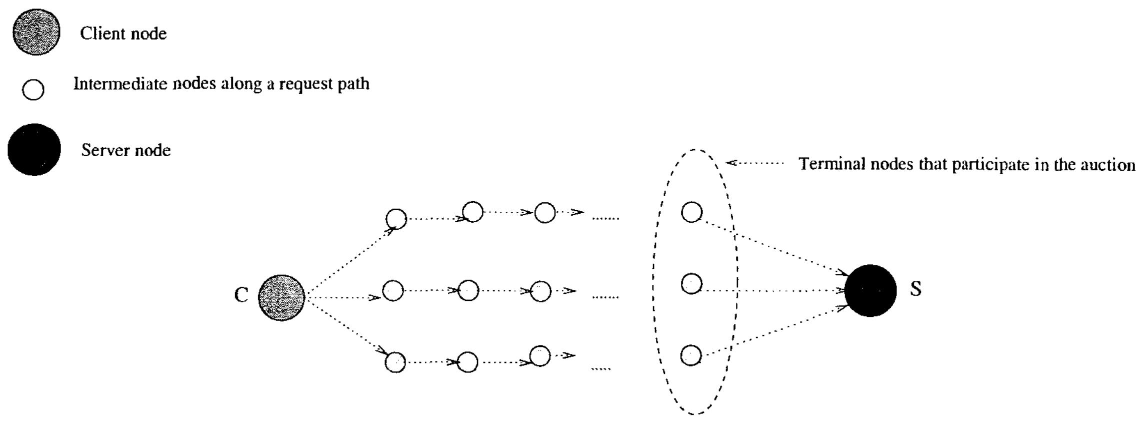

Client node
Intermediate nodes along a request path
Server node
Terminal nodes that participate in the auction
C
S

Figure 4.1 Formation of request chains due to the propagation of lookup requests.

Data is assumed to have utility for nodes and thus have costs associated with them. In other words, unlike in traditional P2P networks [9, 1, 10], where a node can request any number of files, a *client* node in our model is required to pay a reward to the supplier to be able to acquire the desired data. The reward might have to be paid not only to the provider or *server* that supplies the data, but also to the intermediate nodes that assist in locating the server.

For a lookup process initiated by a client, a network can be modelled as comprising of three types of entities - the client itself, the intermediate nodes (including the terminal nodes), and the server providing the data. Nodes incur a cost during a lookup process. The server nodes, especially that serve large sized data, are severely affected under the conditions of heavy load and might be able to serve only some of the requests. Typically, the requests that offer higher prices are given preference over others that offer lower prices. Since, clients incur cost for initiating lookups, we assume that it is in the clients' best interest to successfully obtain the data in as few lookup transactions or attempts as possible.

Unless otherwise specified, all message communication is assumed to provide message non-repudiation. So, for example, if node *1* needs to send a message M to node *2*, then instead of directly sending M, it sends $M^e = (M, T, E(private - key_1; M, T))$ to *2*. $E(private - key_1; M, T)$ is an encryption of M using a well-known algorithm using the public key $private - key_1$ of node *1*. T is the time stamp at which the message is sent. Node *2* after receiving M^e obtains the public key of *1* and verifies that $E(private - key_1; M, T)$ indeed has a true copy of *M* that it receives separately as part of M^e. Node *2* also keeps a copy of $E(private - key_1; M, T)$ that can be used later to prove to anyone that the message *M* was indeed sent to it by node *1*. The intent of including an encrypted copy of the message is not message authentication, but to use it as a proof (if required later) that the message was indeed sent by the sender to the receiver. Our protocol relies on message non-repudiation to ensure that nodes do not go back on their *commitment* as suggested by the contents of the messages sent by them. As explained in Chapter 1, it is assumed that there is a mechanism in place to punish nodes if it can be proven that they did not fulfill their commitments.

Now we discuss how *Problem 2* mentioned in Chapter 1 is dealt with by the incentive-driven lookup protocol.

4.3.1 Problem Formulation

Service type (S): Accessing data in P2P networks.

Problem Definition: Develop a mechanism such that resources like data and bandwidth (used for routing of lookup messages) can be priced in P2P networks.

Service Goals (G_S): We identify the following objectives for the service type defined above.

1. Nodes must be motivated to share data and provide routing service to others.

2. Truth-telling, with regards to the marginal cost of providing data and forwarding a lookup message, should be the best strategy for the data providers and intermediate nodes, respectively. (In P2P networks, peers route lookup messages for each other; the nodes through which a lookup message propagates, are referred to as the *intermediate nodes* for that request).

3. Payoffs received by nodes should be greater than their marginal cost of providing the service, i.e., providing data and/or bandwidth.

4. A client should not be charged an arbitrarily high price for requesting data.

The protocol P_S that we propose for service S provides a mechanism for carrying out lookups and pricing each step of the lookup process, such that nodes have incentive to forward messages and share data with each other. We first describe the proposed protocol, evaluate it against several possible threat models, and then give its implementation on Chord-based P2P networks.

4.4 Description of the Incentive-Driven Lookup Protocol

The proposed protocol correspond to activities in real-world economic markets, where buyers pay money to sellers and intermediaries that facilitate the transactions. However, unlike in the real-world, there are no well established protocols (government rules and policies), and institutions and infrastructure (such as stock exchanges) in a typical P2P setting that can govern the parameters (such as the price charged, the place of occurrence, etc.) of the transactions. Due to such constraints several non-trivial issues need to be addressed - setting resource prices, determining payoffs to intermediate, preventing cheating, etc. We address these issues by the proposed protocol. To simplify our discussion, we consider an example of a lookup process and see how it is carried out under the given protocol.

4.4.1 Parallel Data Lookup

The client C before initiating a lookup for data D, provided by server S, estimates its utility U_D^C of the data. U_D^C is the maximum price that it can offer for data D. C then sends a separate lookup message towards each of the terminal nodes. Together these parallel lookup messages constitute a single lookup process initiated by C for data D.

Each lookup message Msg_{lookup} contains the following information, as included by the client - address of one of the k terminal nodes (T_{D_i}), the data ID (D), the maximum price offered (P_C), the marginal cost (MC_{total}), the request IDs ($Reqid_{private}$ and $Reqid_{public}$).

$Reqid_{public}$ identifies the lookup process such that S (and intermediate nodes) on receiving multiple lookup messages knows that the messages pertain to the same lookup process. Thus, the same value of $Reqid_{public}$ is included in all the lookup messages. On the other hand, a unique value of $Reqid_{private}$ is included in each of the lookup messages. In Section 4.4.5, we illustrate the significance of $Reqid_{private}$. MC_{total} contains C's marginal cost MC_C. Each intermediate node on receiving the lookup message updates MC_{total} by adding its own marginal cost to the received value.

Intermediate nodes for all the lookup messages route the received lookup message to the next hop neighbor (called the *successor* node) and this process continues till the message reaches the desired terminal node. Since the terminal nodes store the index containing the IP address of *S*, they contact *S* in order to obtain the data. *S* receive k such requests and from the $Reqid_{public}$ values knows that the requests pertain to the same lookup process. *S* then holds a second price sealed-bid auction (also called Vickrey auction [54, 24]) with all the terminal nodes as the bidders. *S* provides the data to the terminal node that offers the highest price. The request chain containing the highest bidder, i.e., the winning terminal node, is called the winning request chain *WRC*.

4.4.2 Bidding for the Data By the Terminal Nodes

In Vickrey auction, the highest bidder wins the auction, but the price that it has to pay is equal to the second highest bid. Vickrey auction has several desirable properties, such as existence of truth revelation as a dominant strategy, efficiency, low cost, etc. Vickrey auction in its most basic form is designed to be used by altruistic auctioneers, which are concerned with overall system efficiency or social good as opposed to self-gains. Self-interested auctioneer is one of the main reasons why Vickrey auction did not find widespread popularity in human societies [65].

Since, *S* (the auctioneer) behaves selfishly and tries to maximize its profit, the auction process needs to ensure the following.

- Selecting the highest bidder is the best strategy for *S*.
- The price paid by the highest bidder is indeed equal to the second highest bid, i.e., *S* should reveal true second highest bid to the highest bidder.
- Collusion among *S* and the bidders should not be possible.

In view of the above requirements, we provide a two-phase secure Vickrey auction protocol, which is described in Section 4.4.3. Phase one of the protocol is similar to an earlier protocol in [66] for secure second-price auctions. In both the protocols, bidders initially send encrypted copies of their bids to the auctioneer. In subsequent discussion, we denote the highest and second highest bids by M_1 and M_2, respectively. The price offered by a terminal node to *S* is equal to $P_C - MC_{total}$. The amount of profit made by the *WRC* is equal to $(M_1 - M_2)$. This profit is shared fairly among the nodes of the

WRC (and the client) in proportion to their marginal costs, i.e., nodes with higher marginal costs get a higher proportion of the total profit, and vice versa.

4.4.3 Two-Phase Secure Vickrey Auction to Determine the Data Price

S employs a two-phase Vickrey auction to select the highest bidder and determine the price at which the data is provided.

1. In the first phase, the bidders send encrypted copies ($E(randKey_i; b_i)$) of their bids in message Msg_{bid} to *S*. Here $E(randKey_i; b_i)$ is the encryption of bid value b_i of terminal node T_{D_i} using a randomly chosen secret key $randKey_i$. Each message Msg_{bid} also includes $Reqid_{public}$ value received by a terminal node, so that S can determine that the bids pertain to the same lookup process.

2. The received encrypted bids are sent by *S* back to all the bidders in message $Msg_{bid-reply}$. Since after receiving $Msg_{bid-reply}$, the bidders have encrypted copies of all the bids (total k such bids), *S* is unable to (undetectedly) alter existing or add fake bids.

3. Now each bidder after receiving the message $Msg_{bid-reply}$, sends its secret key in message Msg_{key} to *S*. The received key values are now sent by *S* back to all the bidders in message $Msg_{key-reply}$. At the end of this phase, *S* and all the bidders are able to open the encrypted bids and find out about the highest and second highest bids.

4. In the next and last phase of the auction, S sends a message Msg_{cert} to the winning terminal node ($T_{D_{WRC}}$) certifying that it has won the auction. The received certificate is forwarded along the reverse path, i.e., opposite to that followed by the lookup request, till it reaches *C*.

Msg_{cert} contains the following information - data value denoted by *content(D)*, the highest bid M_1, the second highest bid M_2, the total marginal cost MC_{total} (received by *S* in Msg_{bid}), and the IP addresses of all the terminal nodes that participated in the auction (in Section 4.4.5, we explain how this information is utilized by *C* in order to verify the auction results). The information in messages Msg_{cert} and Msg_{lookup} allow the intermediate nodes, including $T_{D_{WRC}}$, to calculate their reward for being part of the *WRC*. The possession of messages Msg_{cert} and Msg_{lookup} serves as a contract between a node and its predecessor (one which sent the lookup message initially) regarding the reward that the node is entitled to receive from the predecessor. The knowledge of the auction results also enables C to determine the price that it finally has to pay for data D. The calculations of the exact payoff values are discussed next.

4.4.4 Rewarding Nodes of the *WRC*

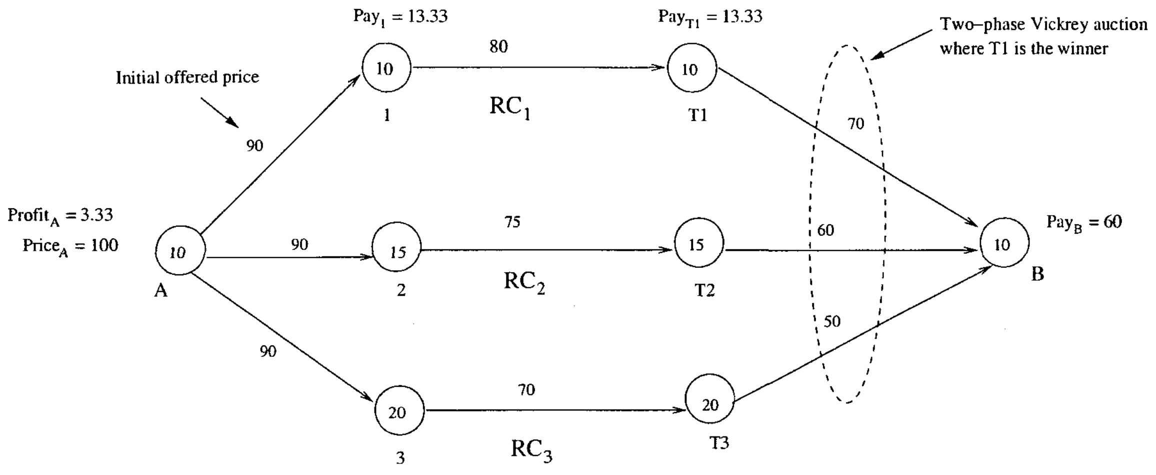


Figure 4.2 A lookup example illustrating how payoffs are distributed among the *WRC* nodes based on their marginal costs.

Msg_{cert} includes the total marginal cost value MC_{total} of all the nodes in the *WRC*. This information along with the highest and second highest bids determine each *WRC* node's payoff. For example, node x's payoff Pay_x is calculated as follows.

$$Pay_x = MC_x + (\frac{MC_x}{MC_{total}} * (M_1 - M_2)) \tag{4.1}$$

The amount received by S is equal to M_2 ($> MC_S$). The profit share of C, i.e., the portion of its initial offer that it saves or gets to keep, is similarly calculated as given below.

$$Profit_C = (\frac{MC_C}{MC_{total}} * (M_1 - M_2)) \tag{4.2}$$

Let us consider a simple example given in Figure 4.2 to better understand the above equations. Three request chains (shown as RC_1, RC_2, and RC_3) are formed as part of the lookup process initiated by node A. Numbers within the circles represent the nodes' marginal costs. *T1, T2, T3* are the respective terminal nodes that store the data index, i.e., they store the IP address of node *B* that owns the desired data. Node *B* on receiving the lookup requests conducts a Vickrey auction, as a result of which *T1* is selected as the winner, but the price it pays is 60. The results of the auction are sent back to node A, and also seen by all the intermediate nodes along RC_1. The resulting payoffs to the intermediate nodes and node *B* are indicated in the figure. For example, payoff to node *1* is 13.33 (=10 + (10/30)*(70-60)). Node A's profit share is 3.33 (= (10/30)*(70-60)). Thus, node *A* effectively has to pay 86.67(=100-10-3.33) for data whose utility to it (after deducting the marginal cost) is in fact 90. Therefore, the proposed scheme based on Vickrey auction ensures that everyone, including the client, server, and intermediate nodes constituting the *WRC* benefit, i.e., earn more than their marginal costs, by participating in the lookup process. This potential of earning higher profits motivate nodes to share their data and forward messages for others.

C after receiving Msg_{cert} determines and takes away its profit share and gives the remainder of its initial offer to the successor node along the *WRC*. The successor node determines its own payoff using Equation 4.1 and after keeping that amount transfers the remaining to its successor, and so on. This process is repeated till the server receives its due payoff. In the above example, node *A* after keeping its profit share (and the amount equal to its marginal cost) gives 86.67 to node *1*, which after keeping its payoff gives 73.34 to *T1*. Now *T1* after keeping its payoff gives the remaining (i.e., 60 ($\approx$ 73.34-13.33)) to node *B*. The amount received by node *B* thus equals M_2 (=60).

A node cannot default on its payment to its successor, since as mentioned earlier, the content of messages (Msg_{lookup} and Msg_{cert}) form a non-refutable contract between a node and its predecessor regarding the amount of money that the node is to receive from its predecessor.

Figure 4.3 summarizes the steps involved in the incentive-driven lookup protocol. The various messages used, along with the information they contain, are also summarized in Table 4.1 for an easy reference.

Step 1: Client initiates the lookup process by sending a lookup message Msg_{lookup} towards $T_{D_i} \forall i \in \{1, \ldots, k\}$
- Intermediate nodes update the value of MC_{total} before forwarding the lookup message
- Lookup messages reach the terminal nodes

/* Vickrey auction - Phase I */
Step 2: Terminal nodes on receiving Msg_{lookup} send Msg_{bid} to the server

Step 3: Server waits for k Msg_{bid} messages (i.e., bids) or till some maximum time τ
- Bids are identified as belonging to the same lookup process by using the value $Reqid_{public}$

Step 4: Server sends message $Msg_{bid-reply}$ to the terminal nodes
- After the above step the bidders have encrypted copies of all the bids

/* Vickrey auction - Phase II */
Step 5: Terminal nodes send their secret key to the server in message Msg_{key}

Step 6: Server replies with a message $Msg_{key-reply}$ distributing the secret keys among the bidders

/* Vickrey auction ends */
Step 7: Server sends message Msg_{cert} to $T_{D_{WRC}}$, which is sent to the client using the reverse lookup path

Step 8: Client (optionally) verifies the auction results by contacting the terminal nodes

Step 9: Reward is given to the nodes of the *WRC* (including the server)

Figure 4.3 Incentive-driven lookup protocol steps

Msg_{lookup}	$T_{D_i}, D, P_C, MC_{total}, Reqid_{public}, Reqid_{private}$
Msg_{bid}	$E(randKey_i; b_i), Reqid_{public}, MC_{total}$
$Msg_{bid-reply}$	$\cup E(randKey_i; b_i), Reqid_{public}$
Msg_{key}	$randKey_i, Reqid_{public}$
$Msg_{key-reply}$	$\cup randKey_i, Reqid_{public}$
Msg_{cert}	$content(D), M_1, M_2, Reqid_{public}, MC_{total}$, IP addresses $\forall T_{D_i}$

Table 4.1 Various messages comprising the incentive-driven lookup protocol

4.4.5 Threat Models

In this section, we evaluate the robustness of the proposed incentive-driven lookup protocol in the face of nodes' selfishness. In particular, we identify and analyze our protocol against potential threat models and show that truthfully following the protocol steps is the best strategy for the selfish nodes.

Threat Model A (Cheating by the auctioneer). Since the auction by *S* takes place in a completely distributed environment, the bidders are unaware of each others' bids and also cannot monitor *S*'s activities. In such a scenario, using the traditional single-step Vickrey auction, where the bidders directly send their bids in clear to the auctioneer, would enable *S* to easily manipulate the auction results. To understand this, let us again consider the example given in Figure 4.2. If traditional Vickrey auction is used, then *B* on receiving the three bids of 70, 60, and 50 knows that *T1*, which is the highest bidder, is willing to pay any amount less than or equal to 70 for data *D*. Therefore, *B* can send a message to *T1* that it is the highest bidder, but the amount it has to pay (i.e., the second highest bid) is 69. Thus, by cheating *B* makes an additional profit of 9.

In order to counter the problem of addition of fake bids (for example, the bid value 69 as explained above) and manipulation of submitted bids by an auctioneer, we use a two-phase Vickrey auction as described in Section 4.4.3. Now the auctioneer, before it can read the bids, has to give encrypted copies of all the received bids back to the bidders. Therefore, in the above example, *B* is unable to send fake bid 69 after finding that *T1*'s bid is 70.

One might argue that there is a possibility for the auctioneer to send different encrypted bids to different bidders if it stands to gain by doing so. However, this strategy would not be effective unless the auctioneer has prior information about the bids it is going to receive, as shown next. Using the same example, let *T1*'s bid be 69, instead of 70 as anticipated by *B*. Therefore, during the first phase of the secure Vickrey auction protocol, *B* encrypts values 50, 60 and 70 and sends it to all the bidders. Here 50 and 60 represent the actual bids and 70 is the fake bid. In the second phase, *T1* (and also *T2* and *T3*) after receiving the decryption keys finds that the highest bid is 70 and that it has lost the auction. Thus, none of the bidders get selected as the winner and *B* by faking the bids gets a payoff of 0 as opposed to 60 that it could have received by not cheating. This observation can be generalized in the form of the following lemma.

Lemma 2. *Assuming a one-shot model of the proposed two-phase Vickrey auction protocol, the auctioneer cannot unilaterally increase its expected payoff by sending fake (encrypted) bids to the bidders.*

Proof. Without loss of generality, suppose the bids are ordered as follows - $b_1 > b_2 \ldots > b_k$. Now the auctioneer can cheat by creating a false second highest bid b_2' (where $b_1 > b_2' > b_2$) and sending it, after encrypting the fake bid with its own key, back to T_{D_1} during the first phase of the Vickrey auction protocol. In this manner, the auctioneer can get a payoff of b'_2 instead of b_2 from the highest bidder. For this cheating by the auctioneer to be successful two conditions must simultaneously be satisfied -

1. The auctioneer should know that the highest bidder (among the *k* bidders) is T_{D_1}.

2. The auctioneer should have precise knowledge about the highest and second highest bids.

However, since the auctioneer has no knowledge about the details of the lookup messages received by the bidders (and their own marginal cost values), and the bids received by the auctioneer are encrypted, the above two conditions for successful cheating cannot be satisfied.

Hence, the auctioneer cannot unilaterally increase its expected payoff by adding fake bids during the two phase Vickrey auction protocol. □

It is possible that even the knowledge of the distribution of the bid values and the number of bids might be exploited by the auctioneer. The auctioneer can send different combinations of encrypted bids to different bidders in order to fake the auction results and thus maximize its expected profit. Such situations are easily handled by the solution proposed for the next threat model. Basically, the strategy is to ensure that the auctioneer is unable to send different bids to different bidders. This is achieved without requiring any costly and difficult to implement communication among the bidders themselves.

Threat Model B (Collusion between *S* and $T_{D_{WRC}}$). The proposed protocol relies on the fact that correct auction results are sent back to *C*, so that the reward is fairly distributed among all the nodes comprising the *WRC*. However, it is possible for *S* and $T_{D_{WRC}}$ to collude and make higher profits by including a fake second highest bid value in Msg_{cert}. For example, in Figure 4.2, by including the value of M_2 as 69 (instead of 60) in Msg_{cert}, *T1* receives the payoff of 79.34 from node *1*, instead of 73.34 that it receives by not colluding. This higher payoff can be shared between *S* and $T_{D_{WRC}}$, and so they both benefit with this collusion.

As mentioned earlier, the message Msg_{cert} sent back to *C* includes the information (i.e., the IP addresses) of all the terminal nodes that participated in the auction. *C* on receiving this information can verify the truthfulness of the received auction results by contacting any (or all) of the listed terminal nodes. These terminal nodes are given incentive to reveal the truth, i.e., disclose the true values of the highest and second highest bid in the auction. The terminal node that identifies that there is a discrepancy (if any) between the auction results received by *C* and the actual values, is referred to as the whistle blower $T_{D_{WB}} \exists WB \in \{1, \ldots, k\}$. *C* can give $T_{D_{WB}}$ part of the money that it saves by detecting the collusion.[1]

A lookup transaction can be considered as a one-shot game in which each participant tries to maximize its profit in a single play of the game. One-shot model is reasonable to assume because the network under consideration is large, distributed, and dynamic. Moreover, it is difficult for nodes to monitor and keep track of others that do not fulfill their collusion agreement. Thus, the terminal nodes have incentive to become a whistle-blower, as they get additional reward from the client (possibly in addition to what they receive from the server for not revealing the truth).

Moreover, *C* upon contacting the terminal nodes ensures that they have the same auction results, i.e., they received the same encrypted bids from the auctioneer during phase one of the auction. Thus, any cheating by the auctioneer, such as sending different encrypted bids to different bidders, as mentioned in Threat Model A, can be easily detected by *C*. This is achieved without incurring excessive message communication overhead required in any bidder discovery and verification protocol, in which bidders identify each other and cross-check each others' bid values.

In effect, *C* acts as a centralized controller for its lookup process and ensures that no cheating by the auctioneer and/or collusion between the auctioneer and winning terminal node takes place. In fact, the proposed solution is effective even if all the nodes (except the client) of the *WRC* collude to get a higher payoff for themselves. This is because the client's profit share is dependent only on its own marginal cost as well as the auction results, which it can verify from the terminal nodes.

Threat Model C (Sending incorrect terminal nodes information). The prevention of collusion in Threat Model B relies on the fact that the information about the terminal nodes sent by *S* back to *C* in Msg_{cert} is correct. However, it is possible for *S* to include fake information about nodes, which it control or with whom it has prior collusive agreement, such that they are guaranteed not to be the

[1]Note that the terminal nodes have verifiable copies of the encrypted bids and corresponding keys that they receive from the auctioneer. This verification is possible due to message non-repudiation.

whistle blowers. *C* will then have no way of cross-checking the bid values and would end up paying more than what it should. To prevent such a possibility, *C* includes a unique request ID $Reqid_{private}$ in each of the lookup request messages it sends. Upon contacting a terminal node, *C* requests the $Reqid_{private}$ value that the node has to make sure that the value is indeed one of the values it initially included in a lookup message. This provides a method for terminal nodes' authentication, as *C* can be sure that it is interacting with a valid terminal node. The results of Threat Models B and C can be generalized in the form of the following lemma.

Lemma 3. *The auctioneer cannot collude with the bidder(s) to increase its expected payoff without being caught by the client node.*

The following theorem can now be stated for the incentive-driven lookup protocol.

Theorem 3. *An auctioneer in incentive-driven lookup protocol cannot cheat.*

Proof. Follows directly from Lemmas 2 and 3. □

Threat Model D (Over-reporting of marginal costs and under-reporting of utility values). We have shown how Vickrey auction can be used to establish utility-driven pricing in a completely untrusted and distributed P2P environments. Vickrey auction results in fair pricing, in the sense that it rewards the client for being truthful in stating its true utility for the data, and also the intermediate nodes for revealing their true marginal costs for forwarding the lookup requests.

An increase in the MC_{total} value for a request chain lowers its final bid, thereby reducing its chances of winning the auction. If intermediate nodes run specialized learning algorithm and gather privileged information about the network state, such as other intermediate nodes' marginal costs that comprise the different request chains, then they may benefit (i.e., make higher profits) by quoting a higher marginal cost and still be part of a *WRC*. Such information, however, is not easy to obtain in highly dynamic environments, and also the information about current network state may not remain valid even in near future periods. In addition, implementing such algorithms can be expensive. Therefore, to minimize MC_{total} it is in each node's best interest to report its true marginal cost.

One may argue that an intermediate node can increase its profit by only slightly increasing its true marginal cost, while not jeopardizing its chances of still being part of a *WRC*. However, we show that in the absence of any privileged information, revealing true marginal cost is the optimal strategy for a node. This result is summarized in the form of the following lemma.

Lemma 4. *Given that a client's utility for a data being looked up is bounded (for example, in Chord [3] if it is less than four times the total marginal cost of nodes comprising the WRC), the best strategy for an intermediate node is to report its true marginal cost while forwarding a lookup request for the data.*

Proof. Let the price offered by bidders (terminal nodes) to an auctioneer (server) are in the interval from $[0, P]$. There are k ($k >> 1$) bidders and we assume that all the k bids are uniformly distributed in the interval $[0, P]$. The average distance between a bid and the next higher bid (assuming that there is one) is $\frac{P}{k+1}$.

Further consider a node, say i, which is part of some request chain and has a true marginal cost of MC_i. For simplicity, we assume that all the nodes belonging to i's request chain have the same marginal cost. (For simplicity, the client's marginal cost is assumed to be zero). We show that if all the other nodes (except i) that are part of the lookup process report their true marginal costs, then the best strategy for node i is to report its true marginal cost only.[2] To understand this, let node i falsely increase its marginal cost by y ($y << MC_i$) and report it as $MC_i + y$ instead of MC_i. Before we proceed further, it is useful to define the following additional variables.

T = total marginal cost of the nodes belonging to the same request chain as node i. T includes MC_i.

τ = price offered by the terminal node of node i's request chain to the server. $\tau \in [0, P]$.

For simplicity, we say that $T + \tau = P$. This is based on the assumption that P represents the client's utility for the data being looked for and that the client uses its true utility value while initiating the lookup process.

Since node i gets a payoff only if the terminal node of its request chain wins the auction, its payoff when it acts truthfully and falsely, represented as E_T and E_F, respectively, are given as follows.

$$E_T = (\frac{\tau}{P})^{k-1} * [\frac{P}{k+1} * \frac{MC_i}{T} + MC_i] \tag{4.3}$$

[2]From Equation 4.1 we know that a node can get a higher payoff by falsely reporting a higher marginal cost value.

$$E_F = (\frac{\tau - y}{P})^{k-1} * [(\frac{P}{k+1} - y) * \frac{MC_i + y}{T+y} + MC_i + y] \tag{4.4}$$

The first term on the right hand side of both the Equations 4.3 and 4.4 describe the probability that node i is part of the *WRC* and the second term gives the payoff that it subsequently receives. We now proceed to show that under the condition when $P < 4 * T$, E_F is less than E_T. For tractability, we further assume a Chord-based P2P network, where the number of neighbors of a node is approximately twice the average hop length of a lookup path. In other words, $T = \frac{k}{2} * MC_i$ (assuming $k \approx \log N$).

E_T is greater than E_F if $E_F - E_T < 0$, i.e., if

$$\frac{\tau^{k-1}}{P^{k-1}} * (1 - \frac{y}{\tau})^{k-1} * [(\frac{P}{k+1} - y) * \frac{MC_i + y}{T+y} + MC_i + y] - (\frac{\tau}{P})^{k-1} * [\frac{P}{k+1} * \frac{MC_i}{T} + MC_i] < 0$$

or

$$\frac{\tau^{k-1}}{P^{k-1}} * (1 - (k-1) * \frac{y}{\tau}) * [(\frac{P}{k+1} - y) * \frac{MC_i + y}{T+y} + MC_i + y] - (\frac{\tau}{P})^{k-1} * [\frac{P}{k+1} * \frac{MC_i}{T} + MC_i] < 0$$

In the above, we use binomial expansion to solve $(1 - \frac{y}{\tau})^{k-1} = 1 - (k-1) * \frac{y}{\tau}$. Higher order terms are ignored, since they anyway further reduce E_F. Solving the above inequality we get,

$$\frac{\tau}{2T} + \frac{k * y}{\tau} < \frac{T}{\tau} + \frac{3}{2} \Rightarrow \frac{\tau}{2T} < \frac{3}{2}$$

Thus, if $\tau < 3T$, we have $E_F < E_T$. This proves that if the client's utility (i.e., P) is less than $4T$, truthfully reporting one's marginal cost maximizes one's (here node i's) payoff. $\square$

Moreover, a client's goal to obtain a data item in a single lookup transaction is best served if it sets maximum possible price for the desired data and that price is the client's utility for the data. The server's marginal cost of serving the request (and of the intermediate nodes) is unknown to the client and can be high depending on the number of requests it is currently serving. Together these factors ensure that using the actual utility value for setting the offered price is the best strategy for the client.

4.4.6 Preventing Free-Riding and Denial-of-Service (DoS) Attacks

The *incentive-driven lookup* protocol provides protection against denial-of-service (DoS) attacks to which other (cooperative) P2P systems are easily vulnerable. In traditional P2P systems, a node can repeatedly make requests for resources without being charged or penalized. Such repeated requests consume network and server resources, and prevent other valid requests from being fulfilled. However, in our proposed protocol such attacks are made costly, and are thus unlikely to be launched by malicious nodes intending to overburden the limited resources of the system. This is because on completion of a successful lookup request, the originating node is required to make payments to other nodes that supplied the data and routed the lookup request. Thus, it is expensive to launch DoS attacks against any server and/or network in general.

By charging the clients for the lookup process, our protocol minimizes the problem of free-riding in the network, as the nodes themselves also have to share their resources in order to be able to pay for the resources they acquire. Nodes are charged for the data they acquire, and the amount charged is at least the sum of the marginal costs of the intermediate nodes that route the request and server that serves the request.

4.5 Implementation of the Incentive-Driven Lookup Protocol Using Chord

We now consider a Chord-based P2P network for describing the implementation of the proposed incentive-driven lookup protocol. We also evaluate the performance of Chord for providing routing service in a network of selfish nodes. We show that in a large network, unless nodes have privileged information about the location of data, following Chord is a good strategy provided that everyone else also follows the Chord protocol.

The incentive-driven lookup protocol described in the previous sections can be implemented as it is on top of the Chord protocol. Only the mechanism by which request chains are formed needs some explanation. It must be noted that now the terminal nodes are the Chord successors of the mappings of a data item onto the Chord network. The term "successor" as used here is the same as used in the description of the Chord protocol, where it referred to the node which immediately succeeds an ID value in a Chord ring or network. The method by which these mappings are determined is explained below.

4.5.1 Data Index Replication

The proposed resource pricing scheme utilizing Vickrey auction is based on competition among different chains of nodes attempting to forward the lookup request and delivering the data back to the client. Higher the competition among the nodes is (i.e., more disjoint the request chains are), higher is the robustness of the pricing scheme. If normal Chord index replication is used, i.e., storing the data index values at the *k* Chord successors of the ID where the data name hashes to, then with high probability, lookups to all these replicas pass through a single node (or a small group of nodes). Such a node can easily control the lookup process and charge arbitrarily high payoffs for forwarding the requests. To avoid such a monopolistic situation and ensure fair competition in setting prices, we propose that data indices be replicated uniformly around the Chord network at equal distances from each other. In other words, data Chord ID mappings should span the entire Chord ID space; this ensures that the lookup paths to different index replicas are maximally disjoint, and are not controlled by any single node. Below we give a mechanism for determining the location for storing index replicas in the network.

If data D hashes to Chord ID DI (i.e., the output of the hash function, whose input ID is D, is DI),[3] then the k data index replicas map to the following Chord IDs.

$$DI_{D_i} = (DI + \frac{2^m}{k} * (i-1)) mod(2^m),\ \forall i \in \{1, \ldots, k\} \tag{4.5}$$

The index values are then stored at the Chord successors (the terminal nodes) of the IDs represented by $DI_{D_i}\, \forall i \in \{1, \ldots, k\}$. The intent of replication in Chord is to simply obtain fault-tolerance, while in our protocol the intent is to obtain both fault-tolerance as well as fair pricing. Uniformly spacing the data indices ensure that the lookup paths for different index copies are as node disjoint as possible. This is evident from the results of Figure 4.4, where we find that the replication strategy described above decreases the probability that the same nodes are included in multiple paths to reach the index replicas. Similar results would be obtained for a network of any size and any replication factor.

[3]The hash function used for computing data Chord ID mapping is the same as that used for determining Chord IDs of the nodes.

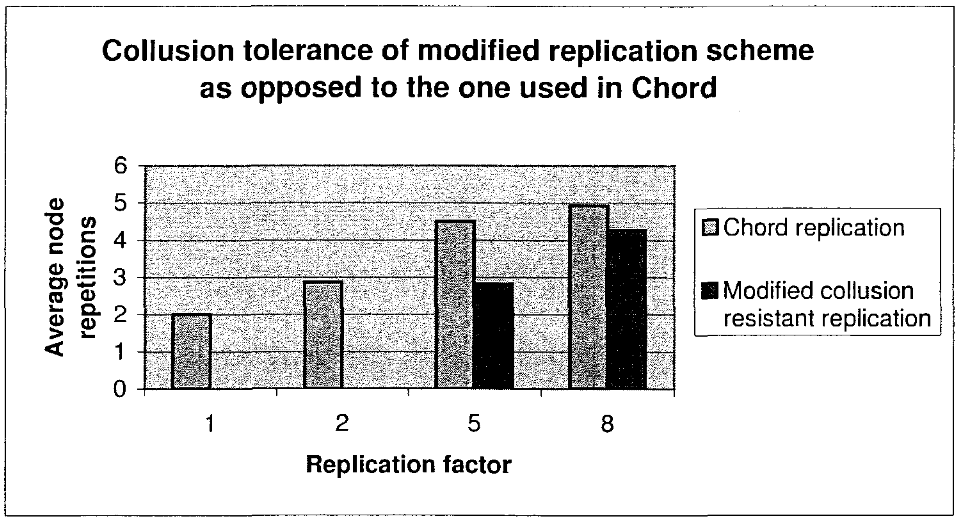


Figure 4.4 Average number of repetitions of intermediate nodes that appear in multiple lookup paths to the data index replica copies. Size of the network $N = 500$

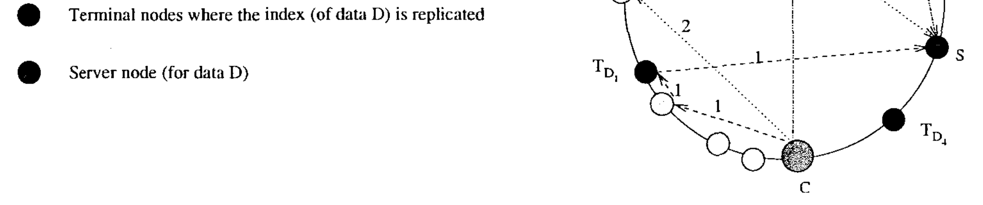

Data index replication factor = k = 4
Size of Chord ID in number of bits = m (=5)
Client node
Neighbors of the client node
Terminal nodes where the index (of data D) is replicated
Server node (for data D)
T_{D_1}
T_{D_2}
T_{D_3}
T_{D_4}
S
C
1
2
3

Figure 4.5 Lookup message propagation in the *incentive-driven lookup* protocol.

4.5.2 Basic Lookup Phase

The lookup phase involves sending lookup messages towards all the terminal nodes of a data item, such that at most one message is sent out for all the terminal nodes that go through the same next hop neighbor - the terminal node selected is one which is closest to that neighbor. For example, in Figure 4.5, *C* sends a single lookup request message towards T_{D_3}, instead of sending towards both T_{D_3} and T_{D_4}. Due to the nature of the Chord routing protocol, with high probability, the number of hops required to go from *C* to T_{D_3} is less than or equal to that required for going to T_{D_4}. As a result, the number of terminal nodes that are contacted during a lookup process may be less than the total number of terminal nodes for a data item. However, for simplicity and without altering the nature of the results obtained, we assume that all the terminal nodes for a data item are contacted and participate in a lookup transaction. Moreover, various request chains formed during the lookup phase would be node disjoint, as required by the system model in Section 4.3. The proof for the same is given in Appendix 8.

4.5.3 Selfish Network Topology

So far we have assumed that nodes form a Chord network and the lookup messages are forwarded in accordance with the Chord routing protocol. Now we investigate how correct is the assumption that nodes truthfully follow the Chord protocol. Since nodes are selfish and join a network in order to obtain data and maximize their profits, the manner in which they select neighbors has a bearing on how successful they are in achieving these goals. This argument definitely holds true for our protocol, since intermediate nodes take their *cut* (equal to their marginal costs) before forwarding a lookup request. Thus, fewer intermediate nodes generally translate to higher profits for the client.

A node can make higher profits by being close to terminal nodes of as many different data items that it requires as possible. However, if the location of those terminal nodes is not known beforehand, then it might be advantageous for a node to greedily choose neighbors around the Chord network distributed at equal distances from each other. This strategy seems appropriate, especially since the data indices are also uniformly distributed around the network. Consider the network shown in Figure 4.6 in which node *1* fill the *m* (= *5*) entries in its routing table as per the greedy routing approach instead. So if node *1* needs to send a message to the terminal node for data *D*, it can do it in two hops as opposed to three hops required using Chord. In general, one can see that in at least half of the cases, when the data to be looked up has Chord ID mapping in region *(a)*, the greedy routing scheme guarantees that the number of hops required to reach the corresponding terminal node is less than (or at most equal to) what is required by Chord. (This assumes that the network size, N, is large and the nodes are uniformly distributed around the Chord network). Even for the other regions, the greedy approach appears to perform comparably to Chord. This is because node *1* always first sends a lookup message to its neighbor that is closest (and with lower Chord ID) to the data Chord ID mapping, and from there on the message is routed using the Chord protocol.

From the above discussion it appears that nodes do not have motivation to follow the Chord protocol, and can make higher profits by utilizing the fact that other nodes follow Chord. In such a scenario the whole routing service would break down, i.e., instead of $O(\log N)$ routing provided by Chord, $O(N/k)$ hops would be required due to the resulting sequential searches for the data indices. However, minimization of the number of routing hops by the greedy approach when everyone else follows Chord is not correct. We prove below that in a large network, on average the performance of greedy routing

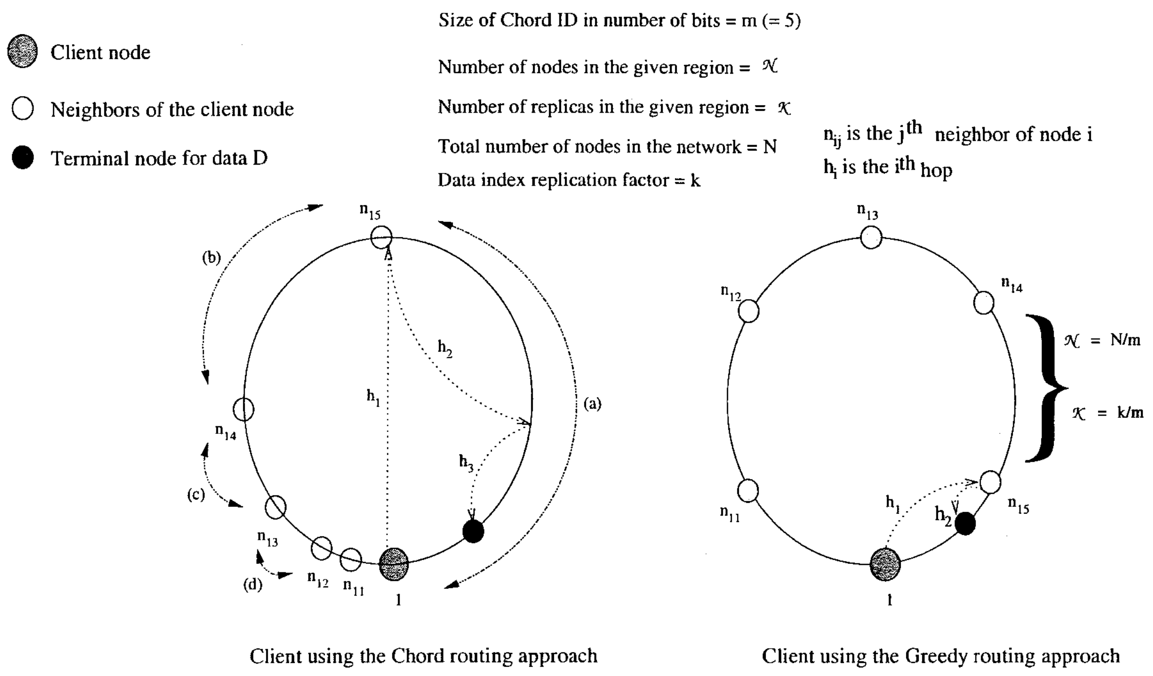

Client node
Neighbors of the client node
Terminal node for data D
Size of Chord ID in number of bits = m (= 5)
Number of nodes in the given region = $\mathcal{N}$
Number of replicas in the given region = $\mathcal{K}$
Total number of nodes in the network = N
Data index replication factor = k
n_{ij} is the j^{th} neighbor of node i
h_i is the i^{th} hop
n_{15}
n_{14}
n_{13}
n_{12}
n_{11}
h_1
h_2
h_3
(a)
(b)
(c)
(d)
1
$\mathcal{N}$ = N/m
$\mathcal{K}$ = k/m
Client using the Chord routing approach
Client using the Greedy routing approach

Figure 4.6 Comparison of Chord and the greedy routing approach. Network nodes are uniformly distributed in the Chord ID space.

approach is no better than that provided by Chord.

Theorem 4. *In a large network, on average the greedy routing approach for any node (say node 1 in Figure 4.6) requires the same number of hops as that required by the Chord routing approach.*

Proof. For the ease of discussion, we assume that the routing table size is fixed for all the nodes in both the approaches and is equal to *m* (same as it is in Chord).

In Chord, the average number of hops required to reach any node from a given node is $1/2 * (\log N)$. Therefore, the average number of hops required to reach any one of the given set of *n* nodes (with Chord IDs in the range $0 - 2^{m-1}$) is $\frac{1}{2} * \log(\frac{N/2}{n/2}) = \frac{1}{2} * \log(\frac{N}{n})$. These *n* nodes are assumed to be located at equal distances from each other. Using these results we obtain the following values.

Average number of hops required by the greedy routing approach to reach one of the k terminal nodes: The neighbors (i.e., the entries in the routing table) of node *1* in this case completely span the entire Chord ID space, i.e., they are uniformly located at equal distances from each other around the Chord network. Therefore, node *1* requires the same number of average hops to reach any of the terminal nodes. (The client first sends the request to its neighbor, which then follows the Chord routing protocol to further route the request).

Thus, the average number of hops taken by the greedy routing approach to reach any data index replica is given as follows.

$$(1 + \frac{1}{2} * \log(\frac{N}{k})) \tag{4.6}$$

Average number of hops required by the Chord routing protocol to reach one of the k terminal nodes: Now we calculate the average number of hops required to reach a data index replica when node *1* (and everyone else) follows the Chord protocol. It will be $(1 + \frac{1}{2} * \log(\frac{N}{k}))$ when the terminal node is located in region (a), $(1 + \frac{1}{2} * \log(\frac{N}{k}))$ when in region (b)[4], and so on (up to *m* such terms).

Therefore, the total average hops needed to reach any of the data index replicas are given as follows.

$$(1 + \frac{1}{2} * \log(\frac{N}{k})), \tag{4.7}$$

which is same as the number of hops given by Equation 4.6.

[4]$(1 + \frac{1}{2} * \log(\frac{N/4}{k/4}))$.

□

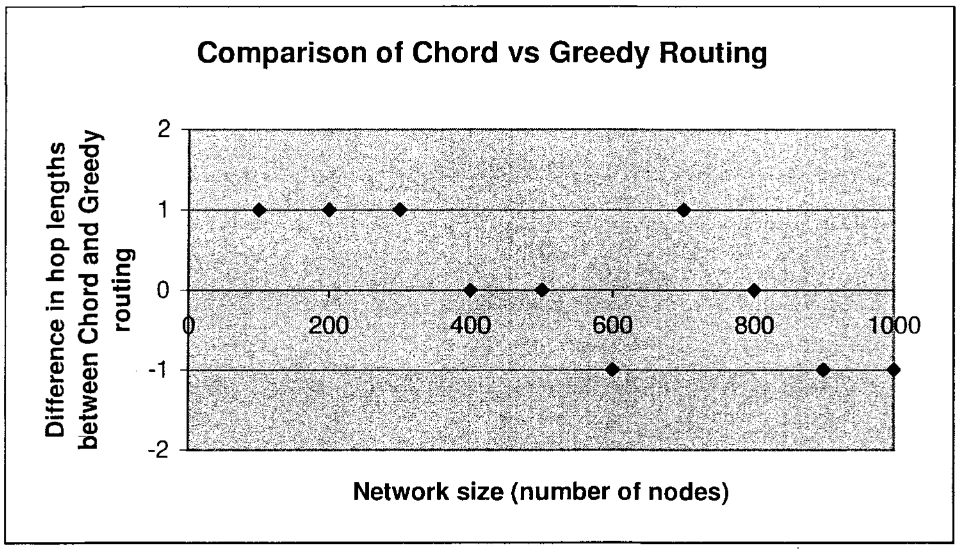


Figure 4.7 Comparison of Chord and the greedy routing approaches with regards to the hop-length.

The results in Figure 4.7 from our simulations also confirm the fact that a node cannot benefit by selecting the greedy routing strategy. Figure 4.7 gives the difference in the observed number of hops when greedy routing is used as opposed to normal Chord routing. We did simulations for varying the number of total nodes and averaged the number of hops for 50 lookups performed for each network size. As can be seen, there is a difference of at most one hop in the two routing strategies and this is true for both small as well as large networks. Thus, a node does not gain an advantage by not following Chord. The following lemma follows directly from this result.

Lemma 5. *If others in a network follow the Chord protocol, it is a good strategy to do the same in order to maximize one's payoff.*

4.5.4 Protocol Overhead

We must admit that the incentive-driven lookup protocol designed to address nodes' selfishness adds some overhead to the system. The overhead is primarily due to two reasons - message communication involved in formation of request chains, including validation of auction results by the client, and the computations involved in message encryption and decryption to achieve message non-repudiation. Note that message non-repudiation is not needed when the client contact the terminal nodes for validating the auction results.

The maximum message processing overhead is incurred by the client and server, but even that is only $O(\log k)$. The number of messages processed by an intermediate node are $O(1)$. The maximum number of nodes involved in a lookup process are $O(k * \log N)$, where k is the number of request chains and $O(\log N)$ is the length of each request chain. Thus, we see that the protocol incurs a reasonable overall message overhead. Also, the protocol deals with the selfishness problem of nodes without relying on any centralized entity or deploying specialized trusted hardware in each network node.

The protocol relies on giving incentives to nodes in order to achieve cooperation in message routing and data sharing. The incentives are typically in the form of reward or money that requires an electronic payment infrastructure, which can be prohibitively expensive. In order to overcome this problem, in Chapter 5, we propose a framework for emulating a system of virtual currency by using reputation as a measure of nodes' wealth.

4.6 Summary

In this chapter, we presented an incentive-driven lookup protocol for searching and trading data in P2P networks. We developed a distributed resource pricing strategy, based on Vickrey auction, to be used in selfish environments, where collusion among nodes is possible. In the process, we addressed a well-known problem of Vickrey auctions - that of a selfish auctioneer, and provided a solution for dealing with it. Our proposed protocol takes selfish behavior of network nodes into account, and ensure that the rewards received by them are maximized if they adhere to the protocol steps. Specifically, the proposed incentive-driven lookup protocol, using the Vickrey auction based pricing strategy, satisfy all the service goals that were identified in Section 4.3.1. Our use of Vickrey auction for resource pricing and handling of *Threat Model D* enforces that truth-telling, with respect to revealing marginal costs of providing service, is the best strategy for the intermediate nodes (service goal 2). Moreover, since payoffs received by nodes are more than their marginal costs, nodes are motivated to serve others, i.e., provide data and forward lookups (service goal 1). Furthermore, the reward distribution mechanism, as described in Section 4.4.4, meets service goals 3 and 4.

We also investigated the applicability of Chord network topology in forming connectivity among selfish nodes. We showed that in the absence of privileged network information, the best strategy for a node is to follow Chord, provided that others also follow the Chord protocol.

CHAPTER 5. A FRAMEWORK FOR REPUTATION MANAGEMENT AND ITS APPLICATION FOR IMPLEMENTING A SYSTEM OF VIRTUAL CURRENCY

5.1 Overview

In recent years, increasingly varied activities are performed on-line, over great distances. Formerly, geographically bound communities are rapidly fused into an expansive collection of world-wide virtual ones. Familiar centralized methods for establishing trust are overwhelmed by the vastness and complexity of open networks. We need to automate methods for trust establishment that are robust enough even when a large fraction of the participants behave selfishly. We believe that future acceptance and success of P2P systems depend mostly on whether or not trustworthiness and reputation of the constituting entities can be established in a reliable and robust manner.

In this chapter, we describe a reputation management framework for large-scale peer-to-peer networks, wherein all nodes are assumed to behave selfishly. By selfish we mean that nodes aim to maximize their own reputation in relation to others in the network. The proposed framework has several advantages. It allows reputation to be used as a measure of nodes' wealth and for emulating a system of virtual currency. The framework is scalable, provides protection against attacks by malicious nodes, and minimizes the problem of free-riding. Nodes can evaluate the reputation of others even without having previously interacted with them. The above features are achieved by developing trusted communities of nodes whose members trust each other and cooperate to deal with the problem of nodes' selfishness, and possible maliciousness.

The concept of currency can be used for designing network operation protocols for various distributed systems [24]. Such systems require giving incentives or payoffs to nodes in order to make them cooperate. Adopting a similar approach, several incentive based protocols exist that reward nodes in the form of increased credit or currency for their cooperation in sharing resources. In all of these currency or "money" is used as a standard tool to achieve the desired network goals. However, a limitation of such schemes is that they either assume the existence of an infrastructure for electronic currency, or some trusted centralized entity that maintain the debit/credit values of the nodes. These assumptions can be difficult to realize in P2P networks.

We suggest that reputation can be used as a basis to provide and procure services. These services can range from sharing data, such as audio/video files, to providing complex distributed processing capabilities as in SETI@Home. In such on-line economies, service providers are suitably compensated for the resources/services they provide. The service providers are willing to serve nodes with higher reputation to increase their own reputation. The earned reputation in turn makes it easier for them to obtain services in future. Thus, the objective of nodes is to maximize their reputation as viewed by everyone else in the system.

Our proposed framework can be used for implementing a system of virtual currencies, which are light-weight, not linked to real-world currencies and which, in fact, are not legal-tender. Since the virtual currencies are not tied to real money, they incur lower transaction costs, and minimize mental decision costs [67] on users for participating in a P2P system.

In order to implement a system of virtual currency, we identify the following minimum guarantees that a reputation management framework need to provide. 1) It should be easier for nodes with higher reputation to access network services. When two nodes contend for a service and only one of them can be served, a node with higher reputation should be serviced. For this to happen the service provider should have incentive to serve the more reputed node. Our framework provides this by ensuring that the reputation of a node increases more by serving a higher reputation node. This is in contrast with traditional reputation management schemes, where service providers make no distinction between the type of nodes (whether of low or high reputation) being served. As an example, say a distributed computing system includes a printer P, and two users A and B. If P receives a printing request from A and B at the same time, but it has resources (paper or cartridge ink etc.) to fulfill only one request, then it is more advantageous for the printer to service the user with a higher reputation. 2) Moreover, like real currency it must be possible for nodes to protect their reputation against attacks by malicious nodes. 3) Furthermore, malicious nodes, either individually or in collusion, should not be able to falsely increase their own reputation.

However, there are important distinctions between real currency and using reputation as its substitute. Real currency can precisely be measured. But reputation is an estimation and different nodes may assign different reputation to a node. Therefore, valuation error is inherent in systems using reputation as a measure of nodes' wealth. Moreover, unlike real currency, reputation gets depleted through expiration and malicious behavior, and so one needs to constantly provide service in order to maintain high reputation. Furthermore, reputation is not necessarily spent or reduced when acquiring service(s), which is not true if real currency is used. This further has the consequence that, unlike real currency, sum of the reputation values of two participants engaged in a service transaction may not be the same before and after the transaction.

5.1.1 Motivation

To the best of our knowledge this is the first attempt in using reputation for implementing a system of virtual currency in P2P networks. In addition to develop this novel application, our research is motivated by several common problems observed in many of the reputation management systems proposed till date. Some of these problems that we attempt to overcome in our proposed reputation management framework are listed below.

- Most of the existing systems rely solely on the positive or negative feedbacks to evaluate and determine the reputation of nodes. The feedback only approach suffers from in-accurate reflection of past experiences of nodes in the respective community and can also be easily manipulated by malicious nodes.

- Most of the existing systems assume feedbacks are honest and unbiased, and lack ability to differentiate feedbacks obtained from less trustworthy nodes and those from trustworthy nodes.

- Most of the existing systems lack temporal adaptivity. They either count all the transactions history of a node without decaying the importance of old transactions in the far past, or only count the recent transactions.

- Several existing systems do not account for the possibility that an entity typically has a group of allies and partners that it trusts more than others.
- None of the systems to the best of our knowledge account for the possibility wherein malicious nodes target specific good nodes to reduce their reputation.

5.2 Related Research

We divide this section into two parts. The first part explores several other reputation management systems that have been proposed for P2P systems. Then in the second part we look at various research projects that attempted to integrate currency into the operations of P2P networks.

5.2.1 Comparison with Other Reputation Management Systems

Several assumptions that are commonly employed in reputation/trust management literature simply do not hold when using reputation as a measure of nodes' wealth. Most importantly, nodes may behave selfishly, and thus the amount of positive information available may be limited. On the other hand negative information may be readily propagated. In addition, nodes have incentive to provide poor recommendations and actively downgrade the reputation of others.

In past several reputation management systems [68, 69, 70, 71, 48, 72, 74] have been proposed specifically for P2P systems. Almost all the proposed schemes assume that majority of the nodes truthfully follow a given protocol, so as to identify the malicious nodes in the system. However, this assumption does not hold true when reputation is also used as a measure of nodes' wealth. This is because everyone now has incentive to behave in a manner so as to maximize their own and minimize others' reputation. Some examples of such a behavior include the following - limiting the amount of positive information regarding others, propagating negative information about others' behavior, giving poor recommendations, etc. A reputation management framework for such systems should therefore explicitly take into account nodes' selfishness, and assume that nodes would follow a given protocol only if it maximizes their own reputation.

Reputation management systems inadvertently rely on recommendations from good nodes to evaluate the trustworthiness of unknown nodes in the network. It is assumed that when a node request its neighbors of their view on some target node, then the majority (if not all) would return their true assessment of that node. For example, in [75] agents when requested sincerely share their views with each other. Such assumptions do not hold true for selfish nodes.

Reputation management schemes, such as [71, 76], do not scale well to large systems. This is because they assume global knowledge and require nodes to store information about every other node in the system. This can be very difficult to implement in large-scale P2P systems.

In a large distributed system, malicious nodes can easily collude and launch complicated hard to detect attacks to break down the reputation management systems. Several earlier proposals [77, 73, 78] do not take into account such threat possibilities.

The reputation ratings generated by our framework identify the degree of trustworthiness of nodes. Proposals such as [72] and [73] consider two and four, respectively, different possible trust levels. This coarse granularity makes it difficult to reason about trust explicitly, as it cannot differentiate between nodes with the same trust level even though they provide different levels of services. Also, the relative reputation rating system of [71] makes it difficult to distinguish good nodes from bad ones.

The authors in [48] develop a trust and security architecture for a routing and node location service that uses a trust protocol, which describes how honest nodes perform. On the other hand, the reputation management framework presented in this chapter is not just limited to enable cooperative routing, but instead is generalized enough to be used irrespective of the nature of services being traded in the network.

The NICE system [74] aims to identify rather than enforce the existence of cooperative peers. It claims to efficiently locate the generous minority of cooperating peers and form a clique of users all of whom offer local services to the community. The system takes a novel approach, rather than using economics to model trust, it proposes using trust to model expected service prices.

5.2.2 Comparison with Other Currency Systems

MojoNation [79] attempted to integrate currency into P2P file sharing. MojoNation was not very successful primarily because it employed a complex and centralized currency model. Moreover, MojoNation focused on file swapping rather than a generic P2P resource market for sharing computational resources.

KaZaA [10], which is currently the most popular P2P system, adopts a currency scheme to incite users to upload files. However, the KaZaA currency is limited to the KaZaA applications. On the other hand, the currency paradigm proposed in this chapter is useful for generic P2P applications.

The KARMA framework [28] uses currency to incite peers to contribute resources. The framework is completely distributed and has been designed to integrate with a structured P2P system. The biggest drawback of KARMA is that no performance results are available regarding its effectiveness and efficiency.

The authors in [26] also proposed a light-weight currency paradigm that allow peers to trade resources and services among each other. The currencies implemented in [26] are not linked to real-world currencies, and are transferrable among applications - that is, an entity can earn currency in the context of one application and spend the same currency in the context of another. Moreover, the paradigm does not impose a single currency in the system, i.e., any entity can issue its own currency. This creates an ultra-free P2P market economy with multiple, competitive currencies - with some currencies having more value than others. The only drawback of the proposed currency system is that it assume that the currency issuers are trustworthy and run on dedicated servers.

The PPay protocol proposed in [80] is secure and efficient, however, it relies on a centralized trusted broker for its correct operation.

5.3 Model Assumptions and Limitations

We assume a P2P network where nodes' wealth is measured by their reputation. Nodes are selfish in the sense that their goal is to obtain desired services and maximize their reputation relative to others. Both the service providers and receivers benefit by obtaining accurate trustworthiness assessment of each other. Service providers want to serve highly reputable clients in order to maximize their own reputation, and service receivers want to maximize the quality of service they receive. The service can be as simple as forwarding a routing message or as complex as downloading data, sending a task for remote execution etc. For concreteness, in the remainder of this chapter, we use file sharing as the example of service being provided in the network.

Each node has a specific area of interest and provides service related to that interest category only. We assume that a node supports at most one service category, and the service category of a node, say *a*, is denoted by $S(a)$. For example, each node download and serve audio/video files belonging to a certain music group or artist only. Service categories themselves are denoted by $S = \{S_1, S_2, \ldots, S_n\}$. The network incurs a cost equal to C_S for completing service $S_i, \forall\, i$. This cost is a well-known parameter, and is due to network bandwidth consumed, buffer space occupied etc. for completing a service. for. For simplicity, C_S is assumed to be constant for all services and/or location of service providers and receivers in the network.

There may be malicious or bad nodes, which provide bad service and/or aim to disrupt the normal operations of a network. Example of malicious activities include spreading viruses, propagating wrong information regarding others, etc. Non-malicious or good nodes, although selfish, do not provide bad service or try to disrupt the operations of a network. Good nodes are differentiated based on the probability, referred to as *service probability*, with which they provide service, if selected by a client. A node either belongs to a set of good nodes (*Good*) or bad nodes (*Bad*). The total number of nodes are denoted by *N* (=$|Good|$ + $|Bad|$).

A node *a* maintains three sets $Good_a$, Bad_a, and ϕ_a, which are the set of good, bad, and unknown nodes, respectively, as perceived by it. ϕ_a represents the set of nodes about whom there is insufficient (or no) information, which allows them to be classified as either good or bad. Initially all the nodes are put in ϕ_a. With time, as more experience is gained about these nodes, their reputation ratings are updated and if appropriate they are moved to either $Good_a$ or Bad_a. For each positive (negative) information that a node receives, it increases (decreases) the reputation of the service provider based on the reputation of the service receiver.

Nodes form trusted communities whose members trust each other to be good nodes and rely on each other for protection from malicious nodes. Such communities are called *trust groups*, denoted by *TGrps*. The members of a TGrp share the same reputation as viewed by the outside nodes. For example, if the reputation of a TGrp with $|TGrp|$ number of nodes is *R*, then the reputation of an individual member node is $R/|TGrp|$. We assume that each node belongs to at most one TGrp. TGrp members are also referred to as *peers*. TGrp of a node *a* is denoted by $aTGrp$ and the same notation is used to denote the set of nodes that comprise this TGrp. Thus, $aTGrp$ is a set of nodes consisting of a and its peers. If two nodes a and b are members of the same TGrp we say $aTGrp = bTGrp$, and if they are not we say $aTGrp \neq bTGrp$.

Reputation of a node is a decaying function of time, i.e., nodes need to continuously share resources and provide services. A node's reputation is incremented only if it serves someone outside its TGrp.[1] If this is not the case then malicious nodes can easily collude and increase their reputation by only serving each other. The reputation of a node is updated after each service transaction, and is represented as follows.

$$R_{provider-new} = f(R_{provider-old}, R_{receiver}) \tag{5.1}$$

Here, f is a function that takes as input the reputation of a service provider ($R_{provider-old}$) and receiver ($R_{receiver}$), and outputs an updated reputation value for the service provider ($R_{provider-new}$). The output value is directly proportional to the reputation of the service receiver. Thus, nodes have incentive to serve reputable good nodes. The above expression holds true for updating a TGrp reputation also, i.e., if node a serves node b, the reputation of *aTGrp* is increased in accordance with the reputation of *bTGrp*.

5.3.1 Service Information Propagation

If a node provides good service then only the network neighbors and TGrp members of the concerned nodes (i.e., the service provider and receiver) become aware of the service transaction. This is because selfish nodes do not propagate positive information in order to prevent others from increasing their reputation. On the other hand, negative information is readily propagated to lower the reputation of the affected nodes. We assume that before a client receives service, it provides information on its neighbors and TGrp members to the service provider. At the end of a successful service completion, the client signs a *satisfaction certificate* (a form of digital certificate) and gives it to the server. The satisfaction certificate is a proof that the server provided good service to the client. The server on receiving the satisfaction certificate sends it to the nodes known to it. These nodes (except the peers), however, do not have incentive to further pass on this positive information. For example, if

[1]As we would see in Section 5.5.2, when a node serves someone in its TGrp, only the peers increase the serving node's reputation.

node a provides good service to node b, the set of nodes that receive this information is given by $aTGrp \bigcup bTGrp \bigcup \mathcal{N}(a) \bigcup \mathcal{N}(b) \bigcup \mathcal{N}(aTGrp)$.

On the other hand, if a server provides bad service, then that information is readily propagated, first by a client to the nodes known to it and then recursively to other nodes known to each of them. As a result, negative information is available to a large fraction of the total nodes and its propagation can be assumed to follow a flooding mechanism. For example, if node a provides bad service to node b, the set of nodes that receive this information is given by V. Although, it is not straightforward for a node that received bad service to prove that fact, it can still send a *complaint* against the server. The complaint message contains digitally signed information about the service provider from which bad service was received. Since, complaints can be easily (even falsely) initiated, it is difficult for the receiving nodes to ascertain the validity of such messages, and consequently reputation of both the nodes (complainant and complainer) is reduced. The decrease in each node's reputation is in proportion to the other node's reputation. This works as a disincentive for selfish good nodes to falsely propagate negative information. Still, sending a complaint against a malicious node does not significantly affect a good node, as malicious nodes typically have low (zero or negative) reputation.

5.3.2 Reputation Management Framework Outline

The reputation management framework presented in this chapter addresses the following issues.

1. TGrps formation
   a. How a node is selected for entry into a TGrp
   b. How a node is evicted from a TGrp
   c. How membership to a TGrp is verified
2. Use of TGrps for reputation management
   a. How positive information about nodes is handled
   b. How negative information about nodes is handled

Simulations are carried out to establish the following properties of the framework.

1. TGrps that provide better service have higher reputation

2. Bad nodes are quickly identified and isolated by good nodes

3. It is difficult for bad nodes to falsely increase their reputation

4. It is difficult for bad nodes to falsely decrease the reputation of good nodes

5.4 Trust Groups

TGrps provide a layer of abstraction such that all the nodes constituting a TGrp are considered equally trustworthy by the outside nodes. Therefore, reputation value is calculated for a TGrp and not for the individual member nodes. This abstraction (although it somewhat reduces the accuracy) provides robustness to reputation estimation due to the following reasons. First, positive information is not widely available, and second, negative information can falsely be initiated and is readily propagated. Due to these factors making an accurate assessment of a node can be extremely difficult. On the other hand, by knowing the reputation of only some members, one can estimate the trustworthiness of other unknown nodes that are part of the TGrp. This is based on the premise that TGrp members behave similarly, i.e., good nodes will include other good nodes only in their TGrp.

Since nodes have limited awareness, the information available at different nodes may be inconsistent. Therefore, formation of TGrps can be a non-trivial task. Bad nodes can also form trust groups, but we reserve the term TGrp to refer to a community of good nodes only.

5.4.1 Peer Selection Criterion

The most important requirement for selecting peers and forming a TGrp is that the constituting members trust each other to be good nodes. This is because the reputation of a node is determined by the reputation of its TGrp, which in turn depends on the aggregate service provided by all the member nodes. Thus, new members are carefully screened before being admitted to a TGrp. Likewise, new nodes carefully select which TGrp (if any) they should join.

Another important criterion for forming a TGrp is to ensure that nodes have no (or minimum) conflict of interest among themselves. Conflict of interest arises if a node suffers a potential loss by increasing the reputation of one of its peers. This can happen, for example, if two peers having the same area of interest contend for a resource. If the resource can be awarded to only one of them, then it is in each node's best interest to not cooperate, and in fact lower the other's reputation. Therefore, we assume that TGrp members belong to different service categories. Moreover, peers do not have any direct incentive to serve each other, because a node's reputation is incremented only if it serve someone outside its TGrp. If this was not the case then malicious nodes can easily collude and increase their reputation by only serving each other.

5.4.2 Trust Groups Formation

In this section, we provide guidelines on how TGrps are formed. The guidelines presented here are used to make our discussion more concrete, and should be used to get a better understanding of the requirements of TGrp formation. Depending on an application scenario additional rules may be used for forming TGrps.

Initially nodes do not belong to any TGrp, i.e.,

$\forall i \in V, iTGrp = \{i\}$ and

$Good_i = \{null\}$, $Bad_i = \{null\}$, and $\phi_i = \mathcal{N}(i)$

Nodes at this time are in an *observation* phase, where they evaluate other nodes based on the information they receive about them. It should be noted that positive information is highly restricted, especially as nodes do not have TGrp members to propagate information for them. Negative information on the other hand is readily propagated and available throughout the system.

Nodes gradually build reputation of others in the network. When the reputation of a node reaches a predefined threshold $R_{threshold}$, that node can be contacted to form a TGrp. $R_{threshold}$ is called the *TGrp entry reputation threshold.* The contacted node agrees to form a TGrp, if it also rates the contacting node highly. The threshold used by the contacted node, say R_{accept}, is equal to (or marginally less than) $R_{threshold}$. R_{accept} is called the *TGrp acceptance reputation threshold.* Both these reputation threshold values are fairly high (set according to system requirements), since it is critical for nodes to have high confidence in the nodes they partner with. The threshold values can be set equal to one's own reputation value, i.e., a good node on finding another node with equal or higher reputation can contact it to form a TGrp. Additionally, both the nodes ensure that they did not receive complaints against each other for a long period of time, called the *observation duration.* The observation duration can be understood to mean the maximum time period when a bad node does not provide malicious service.

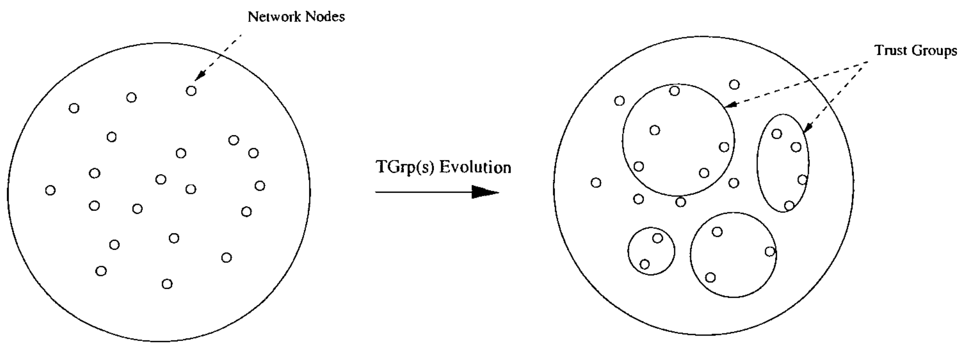


Figure 5.1 Formation of various Trust Groups (TGrps) as nodes gain awareness about each other.

With time several TGrps consisting of small set of nodes are formed. The visibility of nodes increase because they are now also aware of the transactions involving their peers. When the reputation of some other non-member node reaches $R_{threshold}$, it can be invited to join the TGrp. Here $R_{threshold}$ is set equal to the lowest reputation value among all the existing member nodes. The rationale behind this is - if a member node with the lowest reputation value can be included in a TGrp, then an outsider with an equal or higher reputation should also be considered trustworthy enough to be included in the TGrp. The evolution of TGrps is depicted in Figure 5.1.

Figure 5.2 presents the decision rule, referred to as the *join request rule,* used by node *a* to decide whether to ask node *b* to join the TGrp or not.

```
1) if $(b \notin Bad_a) \wedge (R_b \geq R_{threshold}) \wedge (S(b) \neq S(i), \forall i \in aTGrp)$
2) then make a group join offer to $b$; /*if b has high reputation and there is no conflict of interest between it and any of the existing TGrp members*/
```

Figure 5.2 Join request decision rule

The corresponding decision rule, referred to as *join accept rule,* used by *b* to decide whether to accept the join offer or not is given in Figure 5.3.

```
1) if $(i \notin Bad_b, \forall i \in TGrp_a) \wedge (R_i \geq R_{accept}, \exists i \in TGrp_a)$
2) then accept the group join offer;
```

Figure 5.3 Join accept decision rule

In the above figures the service category $S(b)$ of node b is known to node a. This is because node a has information, from the satisfaction certificates, about the service provided by node b in the past. Moreover, $R_{threshold}$ and R_{accept} are calculated as follows.

$$R_{threshold} = min[R_i | i \in aTGrp]$$

$$R_{accept} = R_b$$

The join request decision rule presented above is simplified, in the sense that we omitted consideration of complications that can potentially arise when there is a conflict regarding whether to bring in a new member or not. For example, even though node c might have the same area of interest as node b, still a (where $aTGrp = cTGrp$) might include b in the TGrp (and remove c, if required) because of b's higher reputation. Even though we do not explicitly address such cases to keep the decision rule simple, it is not difficult to see that removing c from the TGrp would require a consensus involving all the peers, because otherwise, it breaks the cohesiveness of the TGrp by alienating the non-agreeing TGrp members. The whole purpose of a TGrp is that nodes within it work in unison and trust each other more than anyone outside the group. Thus, it is reasonable to assume that nodes abide by the above rule for TGrp expansion.

Once a node is admitted to a TGrp, all the member nodes set the reputation of the admitted node to that held by the admitting node.

5.4.3 Trust Group Dynamics

In the previous section, we showed how TGrps are created and expanded. In this section, we look at the reverse process, i.e., how nodes leave or are forced out of a TGrp. Normally, it is difficult

```
1) if $(R_i < 0) \wedge (i \in T)$
2) then evict $i$; /* majority of nodes in $T$ has reputation of $i$ less than 0 */
```

Figure 5.4 TGrp eviction rule

for a malicious node to join a TGrp. A stringent screening function is applied by the member nodes before admitting a new node into their group. Still, it is possible for malicious nodes to join TGrps by providing good service early on till they get admitted to some TGrp. On joining a TGrp they can then provide bad service and/or send false negative information about good nodes. In such situations it is important to identify and remove such nodes from the TGrp. This removal process is also termed as *eviction*. In Figure 5.4, we describe another simple decision rule, termed as *eviction rule*, that is used by nodes of some TGrp *T* to decide if a member node *i* should be evicted.

A member node is evicted only if it provides bad service several times, and its reputation goes below zero as viewed by the majority of the TGrp members. This is because peers trust and give each other benefit of doubt even if complaints are received against them. Several external factors can cause nodes to provide bad service, like broken connection to ISP, network congestion, unintended virus uploads, etc. A member node might also possibly be a victim of false negative information propagated against it by malicious nodes. Thus, member nodes are given a lot of leeway before being evicted from the TGrp. All the good nodes in a TGrp would be in agreement if a particular node needs to be evicted. This is because everyone receive the same negative (and positive) information regarding the target node. Thus, arriving at a consensus for evicting a node should not be difficult.

Evicted nodes are marked by storing information about them. By this we mean that the $R_{threshold}$ values for re-admitting them into the TGrp are suitably increased (based on the network environment and application requirements), say by a factor of 2. Evicted nodes are put in set Bad_i by all the peer nodes. In addition, a node's eviction is made known to as many other nodes as possible. Since eviction signals negative information about a node, propagation of this information follows the same approach as that for normal complaints. Other nodes on receiving the eviction information increase the $R_{threshold}$

value for the evicted node in accordance with the reputation (as viewed by them) of the evicting TGrp.

$$R_{threshold} = R_{threshold} * (1 + R_{evictingTGrp}) \tag{5.2}$$

Here, $R_{evictingTGrp}$ is the reputation of the TGrp from which the node is evicted.

5.4.4 Trust Group Membership Validation

It is important for nodes to be able to correctly prove their credentials regarding their TGrp membership. Moreover, malicious nodes should not be able to fake their membership to some highly reputable TGrp. Furthermore, as nodes join and leave a TGrp, it should be possible to seamlessly update TGrp membership information.

TGrp members create an *affiliation certificate* that include the IP addresses of all the nodes that comprise the TGrp and successively encrypted by each node's private key. For example, say nodes *1*, *2*, and *3* form a TGrp. The IP addresses and private keys of nodes are represented by IP_i and E_i, respectively ($\forall i \in \{1, 2, 3\}$). The following is the affiliation certificate for this TGrp, $E_3(E_2(E_1(IP_1, IP_2, IP_3)))$. One can obtain the public keys of *1*, *2*, and *3* and verify that they all belong to the same TGrp. The affiliation certificate also contains a time stamp such that old certificates cannot be used by (evicted) nodes to falsely claim membership to a TGrp.

A node in order to prove its membership to a TGrp can include the affiliation certificate (containing its IP address as well as that of the TGrp members) when communicating with other nodes. Other nodes on receiving the affiliation certificate can update (if they do not already have) their information about the TGrp. When a TGrp member is evicted, the remaining member nodes create a new affiliation certificate and include it in their message for propagating the information that a node has been evicted. This enables TGrp membership information to be updated by a large fraction of the nodes that receive the new affiliation certificate. Likewise, when a new node joins a TGrp, the updated affiliation certificate includes information about the new node.

5.5 Reputation Management Algorithm

Since nodes behave selfishly and benefit by reducing others' reputation, one cannot rely on recommendations, except from the peers, to evaluate the trustworthiness of nodes. Thus, only direct experience and information received in the form of satisfaction certificates and complaints can be used to construct the reputation. The following - unreliable recommendations, limited reach of positive information and possibility of malicious nodes in large systems, are the main constraints of a reputation management framework for selfish nodes.

In view of the above constraints, we define the following goals for our reputation management framework. With high probability, nodes with higher reputation should get better services, i.e., nodes which provide good service should get good service in return. Moreover, malicious nodes should not be able to fake high reputation and lower the reputation of good nodes in any significant manner, i.e., good nodes should be regarded as good and bad nodes as bad by all good nodes in a system. The chances of a node being identified correctly is directly proportional to how good or bad it is, i.e., its service probability (probability with which a node services a request). These goals are summarized below.

- $Good \nRightarrow Bad_i \; \forall i \in Good$, i.e., good nodes should not be viewed as bad by other good nodes.
- $Bad \nRightarrow Good_i \; \forall i \in Good$, i.e., bad nodes should not be viewed as good by good nodes.

It must be noted that these goals cannot be perfectly achieved in a large and dynamic P2P system with imprecise information. Therefore, the proposed framework strives to achieve these goals with high probability only. In our discussion, all services are considered at par with each other. Thus, the reputation of a service provider depend on the clients it serve and not on the category of service it provides. Similarly, all bad services are indistinguishable from each other, i.e., the reputation of a service provider is reduced based on the reputation of a client that sends a complaint against it.

5.5.1 Reputation Representation and Calculation

A node computes the reputation of every other TGrp that it is aware of. Since the reputation of a node is simply the reputation of its TGrp divided by the number of nodes in the TGrp, we only describe how the reputation of a TGrp is estimated. Nodes that are not part of any TGrp are also considered as TGrps comprising of a single member node. In addition to maintaining the reputation of other TGrps, a node also maintains the reputation of its peers. This is important so as to determine when to add a new member or to remove an (possibly malicious) existing one from the TGrp.

Since the information available at nodes vary, nodes may have different views or reputation of the same TGrp. The consistency in views is proportional to the service probabilities of the member nodes of the target TGrp. A very active TGrp whose member nodes provide service often, is considered good by a large fraction of the nodes. Whereas, a TGrp whose member nodes provide service infrequently, is viewed as good by only a few and unknown by a large fraction of the nodes. Thus, reputation of a TGrp and service probabilities of its members are directly related to each other.

The reputation of a TGrp is always a value between -1 and 1. Higher value corresponds to higher reputation, and vice versa. More precisely, good nodes have positive reputation ratings and bad nodes have negative ratings, and value zero is assigned to nodes in set ϕ. The reputation value is updated after every time period τ, which is a well-defined system parameter. τ can be either timer-driven or event-driven, for example, based on the number of requests observed by a node. A TGrp's reputation is dependent on two factors - number of times its members provided service (and to whom) out of the total service instances in the current time period, and its reputation at the end of the last time period. For illustration, we consider an observer node o and see how it update its reputation of other TGrps (and also its peers). Before that we define the term *network view* that is useful in our subsequent discussion.

Definition 3. *Network view of an observer node, say o, represented by Γ_o, is the set of all nodes that o is aware of. This set will include, o's own ID, its network neighbors, its TGrp members, and other nodes that it becomes aware of upon receiving messages containing satisfaction certificates and complaints. For simplicity, we assume that the size of this set can only grow as o learns about new nodes. In a dynamic system, it is possible for both network neighbors and TGrp members to change. However, as long as these nodes are still part of the network, it is useful to include them in one's network view so as to be able to send satisfaction certificates to them in future.*

For each TGrp, o maintains a reputation counter C_{oaTGrp}, $\forall aTGrp \subseteq \Gamma_o$. C_{oaTGrp} is initialized to zero at the start of every time interval. The counter is increased (decreased) whenever a member of $aTGrp$ provides good (bad) service. Since one gets higher reward for serving more reputable nodes, the increase (decrease) in the counter value is proportional to the reputation of the node being served (cheated). More precisely, if a serves b, where $a, b \in \Gamma_o$ and $aTGrp \neq bTGrp$, o updates $aTGrp$'s reputation counter as follows.

$$C_{oaTGrp} = 1 + C_{oaTGrp} + C_{obTGrp}/|bTGrp| \tag{5.3}$$

At the same time *bTGrp*'s counter is decreased by C_S, which is the cost that the network incurs due to the service transaction and is charged to *bTGrp*. Therefore, we have,

$$C_{obTGrp} = C_{obTGrp} - C_S \tag{5.4}$$

If a cheats b, the counters for both *aTGrp* and *bTGrp* are decremented as follows.

$$C_{oaTGrp} = C_{oaTGrp} - C_{obTGrp}/|bTGrp|$$

$$C_{obTGrp} = C_{obTGrp} - C_{oaTGrp}/|aTGrp| \tag{5.5}$$

For example, say there are three nodes a, b, and c in Γ_o, that belong to TGrps *aTGrp, bTGrp,* and *cTGrp*, respectively. Let the cardinality of all the three TGrps be one. The counter values C_{oaTGrp}, C_{obTGrp}, and C_{ocTGrp} are initially set to zero. o first receives a satisfaction certificate stating that a provided service to b. As a result, C_{oaTGrp} is updated to contain value 1. Subsequently, if c provides service to a then using Equation 5.3, C_{ocTGrp} is updated to value 2 (=1+1). Now, if b is a bad node and sends a complaint against c, C_{obTGrp} is reduced to -2, but C_{ocTGrp} remains unchanged. After these three messages, a and c are put in $Good_o$, and b in Bad_o, respectively. In other words, o views a and c as good, and b as a bad node.

As can be seen, if o receives a complaint message that a cheated b, it decrements the reputation counters for both *aTGrp* and *bTGrp*. This is because complaints can be falsely initiated and it is difficult for o to ascertain their validity. However, it is known that one of the nodes a or b acted maliciously and therefore, the counters for both the TGrps are reduced in proportion to the others' current value. This simple rule ensures that malicious nodes suffer more (incur a larger reduction in counter values) than non-malicious nodes. This is because non-malicious or good nodes provide service to others and typically have higher counter values. This also prevent good nodes from falsely propagating negative information against other (good) nodes. Here we assume that of the two nodes indicated in any complaint, one belongs to $Good$ and the other to Bad.

At the end of the current time interval, o calculates the reputation of all the TGrps in Γ_o using the following equation.

$$R^{curr}_{oaTGrp} = \frac{C_{oaTGrp}}{\Sigma_{iTGrp \subseteq \Gamma_o} C_{oiTGrp}}, \forall iTGrp \subseteq \Gamma_o \tag{5.6}$$

The denominator in Equation 5.6 represents the sum of counter values of all the TGrps that o is aware of. The reputation (R^{curr}_{oaTGrp}) of TGrp, *aTGrp*, obtained above for the current time interval is combined with the TGrp's reputation at the end of the previous interval (represented by R^{prev}_{oaTGrp}) to obtain its new reputation value (R_{oaTGrp}) as shown below.

$$R_{oaTGrp} = \zeta * R^{curr}_{oaTGrp} + (1 - \zeta) * R^{prev}_{oaTGrp} \tag{5.7}$$

Here, ζ is the importance given to the current performance of a TGrp as opposed to its past performance for estimating its reputation. In general, when it is not important (or is implied) to specify the observer node (here o), the reputation of *aTGrp* is simply written as R_{aTGrp}. The reputation of any node, a, represented by R_{oa}, is given as $R_{aTGrp}/|R_{aTGrp}|$. Again, when it is not important (or is implied) to specify the observer node, the reputation of node a is simply written as R_a.

At the end of every time interval, nodes with reputation values below zero are put in Bad_o. If a TGrp has a reputation value of less than zero, then all the member nodes are considered as malicious and put in Bad_o. Specifically, the sets $Good_o$, Bad_o, and ϕ_o are updated at the end of every time period of duration τ, as follows:

$$Bad_o = \{iTGrp | R_{iTGrp} < 0, \forall iTGrp \in \Gamma_o\} \tag{5.8}$$

$$Good_o = \{iTGrp | R_{iTGrp} > 0, \forall iTGrp \in \Gamma_o\} \tag{5.9}$$

$$\phi_o = \{iTGrp | R_{iTGrp} = 0, \forall iTGrp \in \Gamma_o\} \tag{5.10}$$

In the above, we have shown how the reputation of different TGrps (including one's own) is calculated by o. The update of others' reputation as viewed by o can be summarized as given in Figure 5.5.

In the above, we have shown how the reputation of different TGrps (including one's own) is calculated by o. In addition, o calculates the reputation of each of its peers by maintaining a counter C_{oi} for each of them, $\forall i, i \in oTGrp$. These counters are handled similarly to those for the TGrps; their

	$Good_o$	Bad_o	$Unknown_o$
$Good_o$	+	0	+
Bad_o	+	0	+
$Unknown_o$	+	0	+

(a)

	$Good_o$	Bad_o	$Unknown_o$
$Good_o$	–	0	0
Bad_o	–	0	0
$Unknown_o$	–	0	0

(b)

Figure 5.5 Figures (a) and (b) represent the change in reputation of a service provider, as viewed by node a, upon receipt of a satisfaction certificate and complaint message, respectively. Row entries represent the reputation of a service provider, and column entries that of a service receiver, before the receipt of the service transaction message. Here +, -, and 0 represent the increase, decrease, and no change, respectively, in the reputation of the service provider.

exact update mechanism is given in the next section. Peer reputation values are given by the following equations, which are analogous to the ones used above for TGrps' reputation calculation.

$$R_{oi}^{curr} = \frac{C_{oi}}{\Sigma_{jTGrp \subseteq \Gamma_o} C_{ojTGrp}} \tag{5.11}$$

$$R_{oi} = \zeta * R_{oi}^{curr} + (1 - \zeta) * R_{oi}^{prev} \tag{5.12}$$

In Equations 5.6 and 5.11 above, it is assumed that all the reputation counter values are greater than or equal to zero. The reputation counters with negative values are dealt with similarly (and separately).

5.5.2 Reputation Counter Update Algorithm

The algorithm presented here is used by good nodes to update the reputation counters for different TGrps as well as their peers; the reputation values based on these counters are calculated at the end of every time period of length τ, as explained in Section 5.5.1. As before $Good_o$, Bad_o, and ϕ_o represent the set of good, bad, and unknown nodes, respectively, as viewed by an observer node o.

The underlying principle of the algorithm is that a TGrp's reputation is dependent on the reputation of the TGrp(s) it has served (or cheated). Also, reputation ratings are increased by non-TGrp members only if the service provider and receiver belong to different TGrps. We divide the algorithm into two categories - for dealing with positive and negative information, respectively.[2]

Category 1: Node o receives a message (M_{ab}) that a provided service to b. Note that this message is originated by a and contains the satisfaction certificate given to it by b. We use the following notation to describe this event, $o : a \dashrightarrow b$. Based on this information, node o takes an appropriate action as outlined in Figure 5.6.

```
1) if $(b \in Bad_o) \vee (oTGrp \neq aTGrp = bTGrp)$ //nodes a and b are peers, but node o
belongs to a different TGrp
2) then
3)     return;
4) if $(oTGrp = aTGrp = bTGrp)$
5) then
6)     $C_{oa} \leftarrow 1 + C_{oa} + C_{ob}$
7)     $C_{ob} \leftarrow C_{ob} - C_S$
8)     return;
9) if $(oTGrp = aTGrp \neq bTGrp)$
10) then
11)    $C_{oa} \leftarrow 1 + C_{oa} + C_{obTGrp}/|bTGrp|$
12)    $C_{obTGrp} \leftarrow C_{obTGrp} - C_S$
13)    send $M_{ab}$ to $i, \forall i \in \Gamma_o$
14)    return;
15) if $(oTGrp = bTGrp \neq aTGrp)$
16) then
17)    $C_{oaTGrp} \leftarrow 1 + C_{oaTGrp} + C_{ob}$
18)    $C_{ob} \leftarrow C_{ob} - C_S$
19)    return;
20) if $(oTGrp \neq bTGrp \neq aTGrp)$
21) then
22)    $C_{oaTGrp} \leftarrow 1 + C_{oaTGrp} + C_{obTGrp}/|bTGrp|$
23)    $C_{obTGrp} \leftarrow C_{obTGrp} - C_S$
24)    return;
```

Figure 5.6 Reputation counter update upon receiving a satisfaction certificate

Category 2: Node o receives a message (M_{ab}) that a cheated b. Note that this message is originated and signed by b. We use the following notation to describe this event, $o : a \not\dashrightarrow b$. Based on this

[2]In the reputation counter update mechanism presented in Figures 5.6 and 5.7, if a counter value on the right-hand side of an assignment statement is negative, it is set to zero.

information, node o takes an appropriate action as outlined in Figure 5.7.

```
1) if ($b \in Bad_o$)
2) then
3)     return;
4) if ($oTGrp = aTGrp = bTGrp) \wedge (o \neq b) \wedge (o \neq a)$)
5) then
6)     $C_{oa} \leftarrow C_{oa} - C_{ob} * \beta$
7)     $C_{ob} \leftarrow C_{ob} - C_{oa} * \beta$
8)     return;
9) if ($(oTGrp = aTGrp = bTGrp) \wedge (o = b)$)
10) then
11)     $C_{oa} \leftarrow C_{oa} - C_{ob} * \alpha$
12)     return;
13)if ($(oTGrp = aTGrp = bTGrp) \wedge (o = a)$)
14)then
15)     $C_{ob} \leftarrow C_{ob} - C_{oa} * \alpha$
16)     return;
17)if ($oTGrp = aTGrp \neq bTGrp$)
18)then
19)     $C_{oa} \leftarrow C_{oa} - (C_{obTGrp}/|bTGrp|) * \gamma$
20)     $C_{obTGrp} \leftarrow C_{obTGrp} - C_{oa}$
21)     return;
22)if ($oTGrp = bTGrp \neq aTGrp$)
23)then
24)     $C_{ob} \leftarrow C_{ob} - (C_{oaTGrp}/|aTGrp|) * \gamma$
25)     $C_{oaTGrp} \leftarrow C_{oaTGrp} - C_{ob}$
26)     return;
27)if ($oTGrp \neq aTGrp \neq bTGrp$)
28)then
29)     $C_{obTGrp} \leftarrow C_{obTGrp} - C_{oaTGrp}/|aTGrp|$
30)     $C_{oaTGrp} \leftarrow C_{oaTGrp} - C_{obTGrp}/|bTGrp|$
31)     send $M_{ab}$ to $i, \forall i \in \Gamma_o$
32)     return;
33)if ($oTGrp \neq aTGrp = bTGrp$)
34)then
35)     return; /* do nothing - both a and b belong to the same TGrp.
Not much can be derived from this information */
```

Figure 5.7 Reputation counter update upon receiving a complaint

Figure 5.6 is self-explanatory, so we focus on Figure 5.7. α, β, γ (where $0 \leq \gamma \leq \beta \leq \alpha \leq 1$) limit the reduction in reputation counter values for one's TGrp members. Peers are given benefit of doubt even if bad service is received from them. This is because nodes trust their TGrp members to be good and attribute their malicious behavior to some external factor, as described in Section 5.4.3.

For example, in Steps 11 and 15, o minimizes the reduction in reputation counter of its peer node by a factor, α (which is less than 1). Likewise, if a member node receives a complaint involving two of its peers, the reputation counters for both are reduced, but scaled down by a factor, β, as shown in Steps 6 and 7. The value of β is typically less than α because it is not known for certain which of the two nodes is really a malicious node. Thus, a low value of β minimizes wrongfully penalizing a good peer node. Moreover, there is a possibility that none of the two nodes are malicious and poor service occurred because of some external factor. The value γ in Steps 19 and 24 represents the fact that nodes trust their peers more than they trust someone outside their TGrp. Thus, a low value of γ makes it difficult for bad nodes to cause a good node to be evicted from a TGrp by propagating false negative information against it.

In summary, the reputation management framework, as presented above, satisfy the following properties that contribute to its robustness and use for emulating a system of virtual currency.

Property 1 - A node that provide bad service cannot have positive reputation (from Equation 5.5).

Property 2 - Information about bad nodes is readily available (from Section 5.3.1).

Property 3 - A node needs to have positive reputation (i.e., provide good service) to join a TGrp (from join request decision rule in Section 5.4.2).

Property 4 - A TGrp member is not evicted from the TGrp as long as it does not provide bad service (assuming C_S to be zero) (from TGrp eviction rule in Section 5.4.3).

Property 5 - TGrp members are considered more trustworthy than non-TGrp members (from steps 17-20 in Figure 5.7).

Property 6 - Sending false complaints can cause nodes to incur loss in their reputation (from Equation 5.5).

Property 7 - Nodes that are identified as bad are not selected for service transactions by good nodes (from Section 5.3).

Property 8 - Service providers use clients' reputation for prioritizing their requests (from Equations 5.1 and 5.3).

5.6 Framework Evaluation

We have evaluated our proposed reputation management framework using extensive experiments and found that it satisfies the requirements for using reputation as a form of currency, and is robust against possible attacks by malicious nodes. We describe the simulation setup used for our experimentation below.

5.6.1 Simulation Setup

The network contains a specified number of good and bad nodes. Good nodes are differentiated based on their service probabilities, and never provide bad service when selected for a transaction. The goal of bad nodes is to maximize the instances of bad service in the network, i.e., their intent is to get selected for a transaction and then provide bad service. Moreover, they may fake high reputation to increase their priority and prevent legitimate requests from being serviced. Bad nodes may propagate false negative information against a target node to reduce its reputation and cause it to be evicted from its TGrp. Evicted nodes are considered as bad and are given a very low reputation value (say -1) by the peers and others that learn about the node being evicted. We refer to such attacks as the denial-of-reputation (DoR) attacks, and the only way a good node's reputation can appreciably be decreased is by causing it to be evicted from its TGrp.

5.6.1.1 Network Topology and Service Lookups

Nodes have fixed number of network neighbors equal to *log N* connected in an overlay topology, where *N* is the network size.[3] All good nodes belong to one of the specified service categories, i.e., they originate and serve requests only related to that service category. The total number of service categories is dependent on the network size and in all simulations it is set to *N/15*. We also ran simulations with the number of service categories equal to *N/10* and obtained similar results as presented here. Good nodes configure themselves into various TGrps and no two member nodes in a TGrp belong to the same service category. Bad nodes can also join TGrps in order to subsequently provide bad service and/or target good nodes.

[3]This is typically the neighborhood size in several proposed P2P architecture, such as Chord [3] and CAN [4].

Network size	300
α	0.3
β	0.3
γ	0.1
ζ	0.1
C_S	0
Simulation rounds	100

Table 5.1 Default values of the parameters used in simulations.

To keep our framework general and independent of any specific routing protocol, we assume that all nodes belonging to a requested service category are equally likely to be selected by a client. The probability of selection is governed by the service probabilities of the candidate nodes. Moreover, bad nodes can intercept requests and claim to provide service, even if they are not capable of doing so. The probability of interception is directly proportional to the number of bad nodes.

5.6.1.2 Additional Simulation Assumptions

In our simulations, we do not penalize nodes for accessing services, i.e., C_S in Equation 5.4 is set to zero and a node's reputation is unaffected if it accesses service. This is based on the assumption that serving a request is much more expensive than receiving a request and an overloaded node can simply ignore a request if it is unable to serve it.

Unless otherwise stated, Table I gives the value of various parameters used in our simulations. We divide the total simulation time into multiple simulation rounds. In every round, each node initiates a single request that can be satisfied by any of the potential service providers. The limit of one request per node per simulation round simplifies the handling of scenarios where bad nodes repeatedly send out messages to enhance each others' reputation. Intuitively, if one receives multiple service transaction information initiated by the same node in a short span of time, then this information is discarded.

The time interval for reputation update, as defined in Section 5.5.1, is equal to one simulation round. This was adopted so as to simplify the simulation code and any other timer value could also be used without altering the qualitative nature of the results.

5.6.2 Effect of TGrp Formation Rules

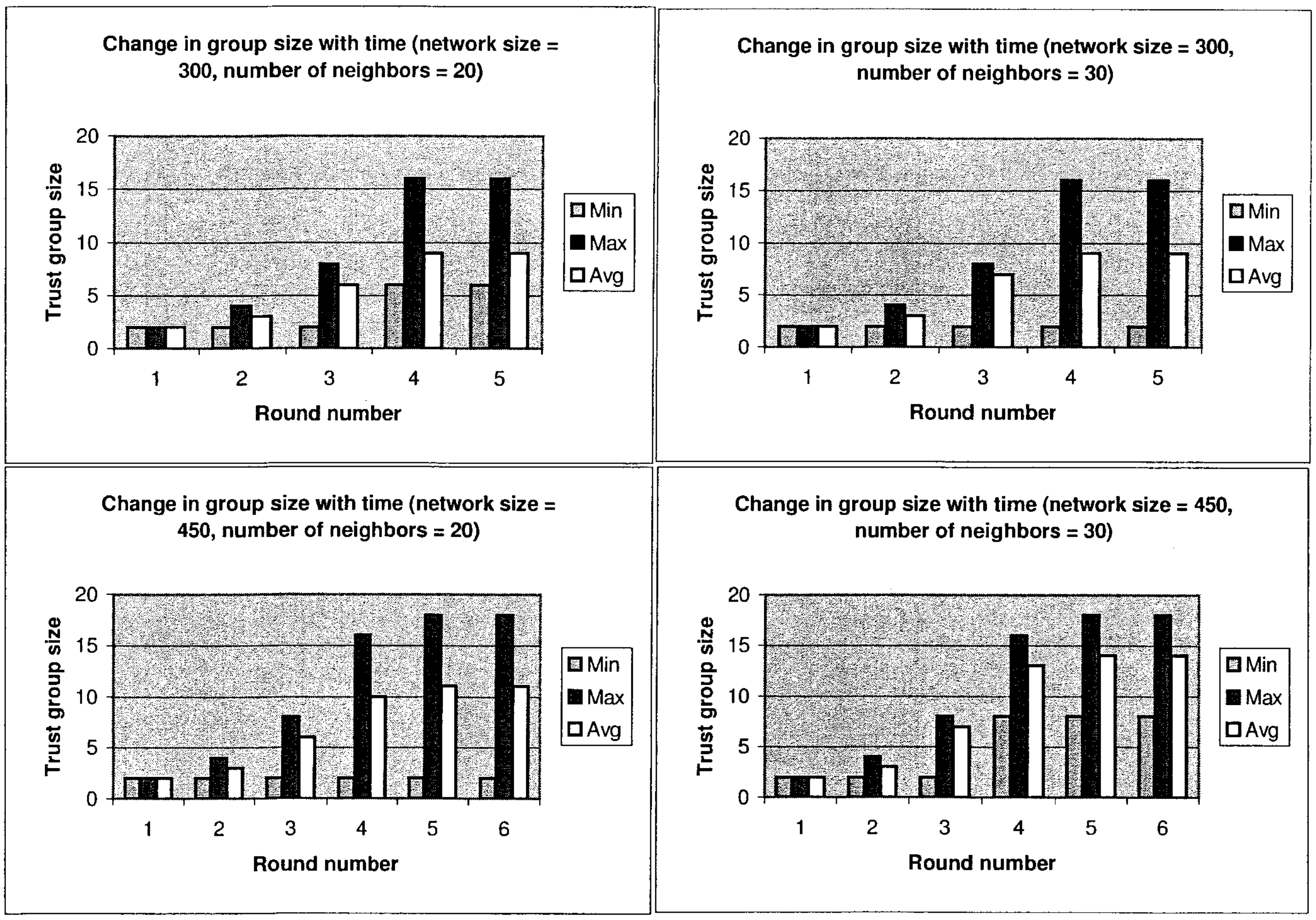


Figure 5.8 Figure depicting the evolution of TGrps in networks of size 300 and 450. For each network size simulation results are given when the number of neighbors of a node were 20 and 30 each. Min, Max, and Avg refer to the minimum, maximum, and average size of the TGrps in the network at the end of each simulation round.

We carried out simulations to study the effect of TGrp formation rules given in Section 5.4.2 to see if an uninitialized system converges to a stable state, such that the number of TGrps (and their respective size) in a system do not change. (During our simulations we allowed TGrps to merge provided there were no service category conflicts among the joining nodes). Figure 5.8 indicate that an uninitialized system reaches the stable state in only a few simulation rounds (in round number 4 and 5 for network size of 300 and 450, respectively). A simple argument to support this fast convergence is that a TGrp cannot have members with the same service category, and thus the number of different service categories limit a TGrp size.

The problem of TGrp formation is akin to formation of communities in human societies. Researchers in social sciences and related areas have proposed models to predict this phenomenon [81, 82]. We believe that these models for community formation complement our work and can potentially benefit from the proposed framework. For the remainder of the chapter we assume the TGrps to be given with all good nodes belonging to one of them. Initially, the bad nodes are not part of any TGrp, however, they can join one if they provide sufficiently good service over a period of time. This scenario is valid in real-world also, as malicious nodes usually do not join a network in the early stages of its formation. Usually it is much easier for malicious nodes to operate when the network has reached a certain critical size.

5.6.3 Effect of Service Probability

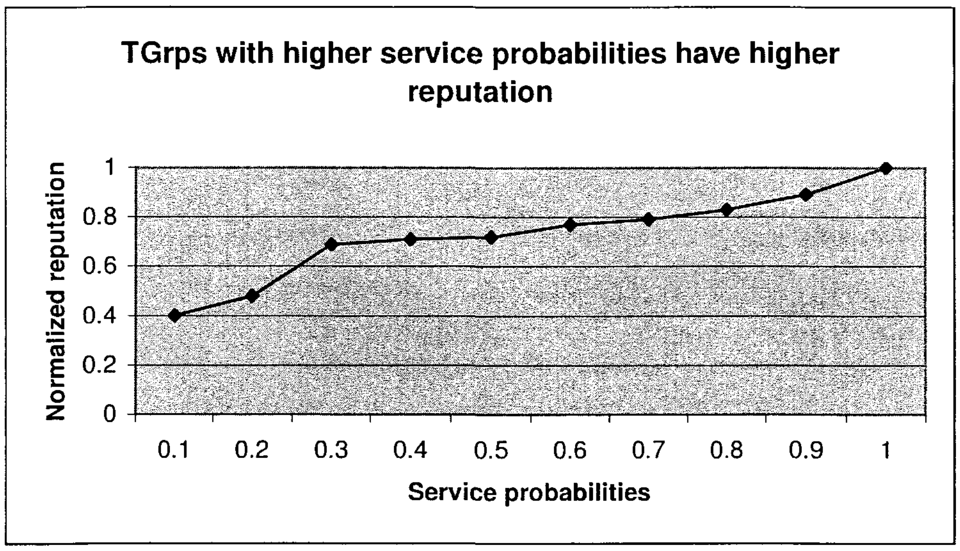


Figure 5.9 Simulation results showing the normalized reputation (scaled to a value between 0 and 1) of various TGrps at the end of round number 50. The network consisted of 10 TGrps, each containing 15 member nodes. Nodes in a TGrp serve with equal probability. The service probabilities of nodes in different TGrps are different and are assigned one of the values as shown in the figure.

Simulation results show that reputation of (and extent of awareness about) TGrps is dependent on the service probabilities of their respective member nodes. This can be seen from the results shown in Figure 5.9. Moreover, with time all bad nodes are detected and isolated (not selected for service transaction), thus limiting the amount of bad service they can provide to good nodes, as demonstrated by the following set of attack models.

5.6.4 Attack Models

To take into account different possible strategies of the bad nodes, we examine five different attack models in this section. These attack models reflect the main requirements outlined in Section 5.1 for using reputation as a substitute for currency, and also indicate the robustness of the proposed framework in dealing with nodes' selfishness and maliciousness.

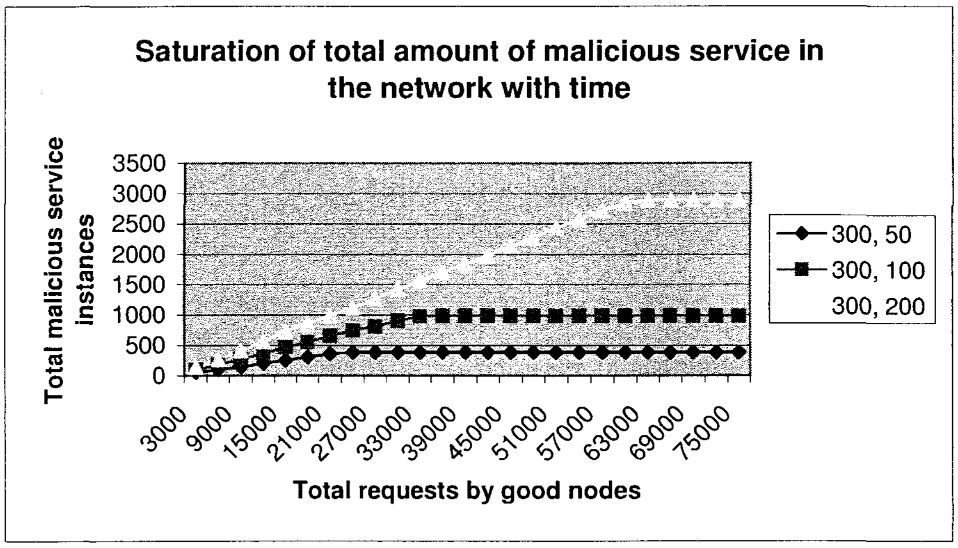
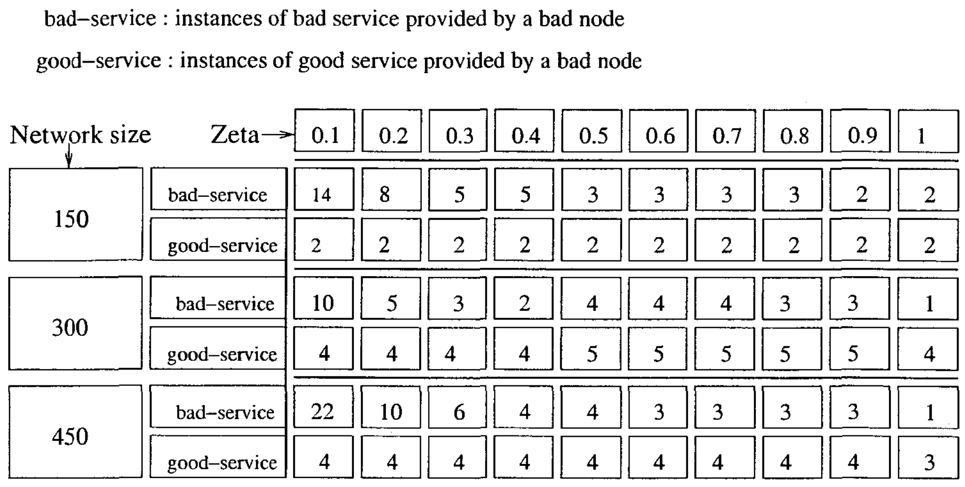


Figure 5.10 Even when there are large number of malicious nodes in the system, the percentage of bad service in the network continuously decrease as the number of requests increase. Bad nodes are identified and not selected for future service transactions. The simulation results are for a network with 300 good nodes, with increasingly higher number of bad nodes. For example, (300, 50) denotes the network configuration with 300 good nodes and 50 bad nodes.

Attack Model A (Always provide bad service). Malicious nodes always provide bad service when selected for a transaction. The probability of a bad node selection is directly proportional to the number of bad nodes in the system. This attack model is countered due to Property 1, Property 2, and Property 7. In this case, the reputation ratings of bad nodes remain zero or negative, and thus they are not able to join any TGrp. Since negative information is readily propagated, all bad nodes are rapidly identified and put in Bad_i set of all the good nodes. Since the nodes in Bad_i are not selected for service transaction, the affect of bad nodes on good nodes is minimized and the network soon continues to operate as if there are no malicious nodes.

As shown in Figure 5.10, the number of bad service instances saturate to some maximum value after only a few simulation rounds. As expected, the time to reach this maximum value is proportional to the fraction of bad nodes in the system.



bad–service : instances of bad service provided by a bad node

good–service : instances of good service provided by a bad node

Network size	Zeta→	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
150	bad–service	14	8	5	5	3	3	3	3	2	2
	good–service	2	2	2	2	2	2	2	2	2	2
300	bad–service	10	5	3	2	4	4	4	3	3	1
	good–service	4	4	4	4	5	5	5	5	5	4
450	bad–service	22	10	6	4	4	3	3	3	3	1
	good–service	4	4	4	4	4	4	4	4	4	3

Figure 5.11 With increasing ζ, the amount of good service provided by a malicious node to join a TGrp exceeds the amount of bad service it can provide before it is evicted from a TGrp. *Network size* refers to the total number of nodes in the system.

Attack Model B (Provide good service until one joins a TGrp). Malicious nodes first provide good service (with service probability equal to unity) and upon joining a TGrp, they always provide bad service. The intent of bad nodes here is to utilize the fact that the reputation of a node is dependent on the reputation of its TGrp. By joining reputable TGrps, malicious nodes attempt to maximize the instances of bad service in the network. This is because bad nodes are not put in Bad_i, as long as good service by their peers offset their bad service. As a result, bad nodes despite their poor service record continue to get selected for service transactions.

This attack model is countered due to Property 3, and Property 4. As described in Section 5.4.3, the reputation of a member node that provide bad service is gradually reduced and is eventually evicted when its reputation goes below zero as viewed by the majority of its peers. Moreover, re-entry of an evicted node into the same or any other TGrp requires higher level of service than what was required of it previously. Therefore, we only consider the damage that a bad node can cause by joining a TGrp for the first time.

We ran simulations to see the total number of bad service instances that a malicious node can provide upon entering a TGrp and before it is evicted. As shown in Figure 5.11, the amount of bad service that a malicious node can provide, before it is evicted, is greatly influenced by the value of ζ. For small values of ζ, the number of bad service instances far exceed the number of good service instances. But as ζ is increased, the reverse is true and this attack model becomes less and less attractive for the bad nodes. Bad nodes need to provide good service to be able to join a TGrp. These results are as expected, since higher ζ means that greater importance is given to current as compared to past performance. Therefore, malicious nodes cannot rely on past reputation to continue providing bad service and still be part of a TGrp.

We believe that it is difficult to completely avoid the damage caused by this attack model. This is due to the premise on which the notion of TGrps is based. TGrp members trust each other to be good and give each other benefit of doubt even if they provide bad service. However, in all our simulation runs, bad nodes are eventually identified and evicted.

Attack Model C (Spread false negative information regarding good nodes). Malicious nodes launch DoR attacks to reduce the target nodes' reputation and cause them to be evicted from their TGrps. This attack model is countered due to Property 5, and Property 6. It requires several bad nodes to provide sufficient good service to cause a target good node to be evicted. This partly defeats the goal of malicious nodes to maximize the instances of bad service in the network. We ran simulations with network size equal to 150, 300, and 450 and in each configuration successively varied the number of bad nodes. All the bad nodes target a randomly chosen good node and spread false negative information against it in each round. Also, in order to earn high reputation, bad nodes provide good service whenever they are selected for a service transaction.

round # : Simulation round number in which the target node is evicted

good–service : instances of good service provided by bad nodes (till the target node is evicted) in order to earn high reputation

Network size	|Bad| →	5	10	15	20	25	30	35	40	45	50
150	round #	–	–	–	34	27	25	23	22	22	22
	good–service	–	–	–	240	276	313	330	374	470	490
300	round #	–	–	–	–	32	31	29	27	25	24
	good–service	–	–	–	–	263	298	314	322	377	384
450	round #	–	–	–	–	38	37	29	28	28	27
	good–service	–	–	–	–	298	323	361	375	403	459

Figure 5.12 With the increase in the number of bad nodes, the target node is evicted in an earlier round. At the same time, however, the total instances of good service provided by bad nodes also increase. Entries left blank indicate that bad nodes were unable to evict the target node, i.e., the DoR attack failed. *Network size* refers to the total number of nodes ($|Good| + |Bad|$) in the system.

As shown in Figure 5.12, in all the simulation runs, at least 20-30 bad nodes are required to cause a target node to be evicted. This is expected because peers of the target node scale down the reputation counter values of the complaining nodes by a factor, γ (=0.1). Thus, large number of complaints are needed before the reputation of the target node goes below zero as viewed by its peers. With higher number of bad nodes the target node is quickly evicted, i.e., in an earlier round, but the total instances of good service provided by the bad nodes is also increased.

Attack Model D (Piggyback on a TGrp's reputation to increase the reputation of other bad nodes). Bad nodes split themselves into two groups - $Bad1$ and $Bad2$. Nodes in $Bad1$ always provide good service and nodes in $Bad2$ always provide bad service. $Bad1$ nodes assist $Bad2$ nodes to increase their reputation. This attack model attempts to maximize the number of bad service instances by increasing the probability that $Bad2$ nodes are selected for service transactions. $Bad1$ nodes initially provide good service in order to be accepted into some TGrp. After joining a TGrp, they do not serve good nodes and only propagate positive information about *Bad*2 nodes. Since *Bad*1 nodes never provide bad service and because C_S is set to zero in our simulations, *Bad*1 nodes never get evicted from their TGrp(s). *Bad*2 nodes on the other hand use the increased reputation to get selected for service transactions and provide bad service.

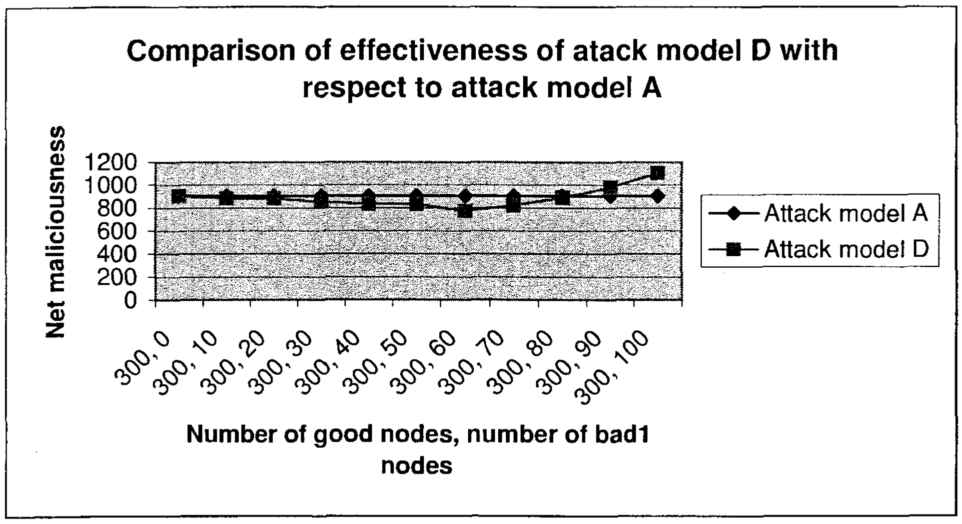


Figure 5.13 Comparison of Attack Models A and D when the number of good nodes are 300 (i.e., $|Good| = 300$) and the number of $Bad2$ nodes are 100 ($|Bad2| = 100$). The number of $Bad1$ nodes vary from 0 to 100 in Attack Model D, whereas they remain equal to 0 in Attack Model A.

This attack model is countered due to Property 1, Property 2, Property 3, and Property 7. To evaluate its effectiveness, we compare it with Attack Model A to see if *Bad*1 and *Bad*2 nodes together are able to increase the instances of bad services beyond what is possible in Attack Model A. For this we define a metric, *net maliciousness*, which is the difference between the number of bad service instances provided by *Bad*2 nodes and good service instances provided by *Bad*1 nodes.

It is clear from the results in Figure 5.13 that the value of net maliciousness in Attack Model D is less than that in Attack Model A, even when there are significantly large number of *Bad*1 nodes. For experimentation, in Attack Model A, we always have the number of *Bad*1 nodes to be zero, while in Attack Model D, we successively increase the number of *Bad*1 nodes to see how it increases the probability of *Bad*2 nodes getting selected for service transactions. When the number of *Bad*1 nodes increase relative to the number of good nodes, $Bad1$ nodes do not have to provide as much good service to get accepted into some TGrp. This explains higher net maliciousness achieved by Attack Model D as compared to Attack Model A when $|Bad1| \geq 90$. Similar results were obtained for other network sizes as well, and it appears that this attack strategy is not any stronger than the one adopted in Attack Model A.

|Bad|	10	30	50	70	90	110	130	150
round #	43	46	45	47	47	51	53	53

Figure 5.14 Illustration of the number of rounds in which the bad nodes are identified as bad by the good nodes. The number of good nodes are 300 and the number of bad nodes vary from 10 to 150. In every simulation round each bad node sends a satisfaction certificate with another randomly selected bad node as the service provider. The service probabilities of the bad nodes are set to unity so as to simulate their increased reputation, and hence higher probability of getting selected by good nodes for service transactions.

Attack Model E (Spread false positive information regarding other bad nodes). Malicious nodes send out false satisfaction certificates, indicating other malicious nodes as the service provider, in order to increase each other's reputation. In addition, they provide bad service when selected by good nodes for service transactions. Malicious nodes can use the increased fake reputation to increase their priority and prevent other legitimate requests from being serviced. However, this attack strategy is naturally countered due to Property 5 and Property 7. An increase in the reputation of malicious nodes attract service requests from the good nodes. But since malicious nodes provide bad service to good nodes their reputation goes down. This decrease in reputation can be substantial depending on the value of parameter γ. In fact, the simulation results confirm this hypothesis, as it is found that all the malicious nodes are identified as bad by the good nodes (see Figure 5.14). Interestingly, number of malicious nodes had little impact on the effectiveness of this attack model. This is primarily due to the limit imposed on the number of requests that a node can originate in any given round.

5.6.5 Minimizing Free-Riding

The framework reduces the problem of free-riding because service providers serve clients based on their reputation, and the only way to increase reputation is by serving others. It is possible for nodes to enter a TGrp and piggyback on the reputation of their peers, however, to join a TGrp one has to first provide good service to sufficient number of nodes in order to earn high reputation. Even after entering a TGrp, free-riders cannot continue to access services without sharing their own resources in return. Peers decrement the reputation counter of a node accessing service by the network cost, C_S. With time, the reputation of a node that only receive services goes below zero and is evicted from the TGrp.

Moreover, service providers serve a client only if the client is not in their Bad_i set. This is because, even if there is no contention for service (as given in the example in Section 5.1), a service provider's reputation is not increased if it serves a malicious node. Thus, it is difficult for malicious nodes with low reputation to receive any service.

5.6.6 Aligning Virtual Currencies With Fine-Grained Monetary Transactions

The reputation management framework presented so far allows reputation to be used as a measure of nodes' wealth, and to ensure that nodes with higher wealth get priority (say, in terms of quality of service received) over others with less wealth. The framework can also be easily extended to more closely mimic the actual monetary transactions taking place among nodes. For example, say node a serves node b in return for Φ amount of currency. This information can be encoded in the satisfaction certificate propagated by node a, and accordingly Equations 5.3 and 5.4 would be modified to include the amount value of Φ. The modified equations are given as follows.

$$C_{oaTGrp} = 1 + C_{oaTGrp} + g(\Phi) * C_{obTGrp}/|bTGrp| \tag{5.13}$$

$$C_{obTGrp} = C_{obTGrp} - g(\Phi) * C_S \tag{5.14}$$

In the above equations, $g()$ is a function that suitably scales up (or down) the monetary amount values to be useful for reputation calculation. One must take care in selecting the function $g()$ to ensure that the increase in reputation (or currency) of the service provider is never less than the decrease in reputation (or currency) of the service receiver. This is because otherwise the total curency in the system would gradually reduce, and eventually become negative.

We believe that the qualitative nature of the results given previously reagrding the effectiveness and robustness of the proposed framework would not change due to the above changes in the calculation of reputation counter values. As part of our future work we would investigate more formally the financial stability (and the accompanying inflation/deflation) of the system using our proposed virtual curency implementation.

5.7 Summary

In this chapter, we proposed and evaluated a novel reputation management framework for large Internet-scale P2P systems. We demonstrated the mechanisms for and feasibility of using reputation as a substitute for actual currency, and described how reputation computation can be carried out when nodes behave selfishly (and even maliciously). The framework implements a system of virtual currency such that nodes with higher reputation stand better chances for obtaining services than their less reputable counterparts. We considered various strategic attack models that can be used by malicious nodes, and showed that our proposed framework gracefully withstand all such attacks. The framework also minimizes the problem of free-riding.

Our most important contribution is to utilize the concept of TGrps for reputation management. We find that even simple reputation update rules, based on the notion of TGrps, are effective when there are large number of malicious nodes working in collusion to bring down the system. The various advantages of TGrps are summarized below.

- Members of a TGrp cooperate to minimize the damage that malicious nodes can cause to their reputation. This is because a node's reputation is derived from its TGrp's reputation, which in turn is dependent on the aggregate service provided by all the member nodes. Thus, even if malicious nodes target a good node by sending false complaints against it, the reputation of the target node is not severely affected.

- TGrp members assist in positive information propagation, i.e., when a member node provides good service, its peers help to propagate that information to as many other nodes as possible. This is because doing so increases the reputation of the entire TGrp, and thus of all the constituting member nodes.

- TGrps provide scalability to the reputation management system because nodes are judged based on their TGrps. Therefore, one needs to keep reputation information about TGrps only, which are much less in number than the total number of nodes in a network.

- Recommendations from peers are more reliable. This is because peers have different areas of interest (and thus minimum conflict of interest), and so have little incentive to provide misleading recommendations.

- TGrps provide an effective solution for dealing with large group(s) of colluding malicious nodes and providing protection against possible attacks by malicious nodes. This satisfies one of the requirements outlined in Section 5.1 for implementing a system of virtual currency - protecting one's wealth against malicious nodes in the system.

Our framework is extensible and flexible enough to be tuned as per the requirements of a specific system. The proposed framework is scalable to large-scale P2P systems, and enables reputation computation in the face of nodes' selfishness, which can cause wrong or no service information to be propagated.

CHAPTER 6. GAME THEORY AS A TOOL TO STRATEGIZE AS WELL AS PREDICT NODES' BEHAVIOR IN PEER-TO-PEER NETWORKS

6.1 Overview

Free-riding is widely acknowledged to be plaguing the current growth and widespread deployment of P2P systems. In [21] it is mentioned that almost 70 percent of the nodes in a Gnutella system never share their resources. Several mechanisms have been proposed to address this problem of free-riding - (1) monetary payments (service providers get suitably compensated by service receivers), and (2) reputation schemes (nodes with higher reputation get better service from others). The monetary payment scheme involves a fictitious currency, and requires an accounting infrastructure to track various resource transactions, and charges for them using micropayments. While the monetary scheme provides a clean economic model, it is difficult to implement such schemes in practice. The reputation based incentive model seems more promising.

In this chapter, we study the behavior of nodes in peer-to-peer networks when reputation is used as a mechanism to incentivize nodes to share resources and provide services. The probability of a node obtaining service is directly proportional to its current reputation, and the only way to enhance reputation is by serving others. This minimizes the problem of free-riding without relying on any centralized entity and/or coordination among peers. Coordination among peers is not required since peers decide their optimal strategies independently. The strategies used are symmetric, i.e., the same strategy is used by all the nodes (as opposed to protocols, which require different nodes to behave differently - this is difficult as all nodes in a P2P system assume the same role, thereby complicating different strategy assignment to different nodes).

Currently, P2P systems are used primarily for file sharing such as audio, video, etc. However, it is widely acknowledged that other resources, such as compute power, bandwidth, storage, etc., can also be potentially shared using the P2P paradigm. Moreover, it is predicted that in future large ad-hoc grids would be organized in a P2P configuration to execute large-scale complex applications [37]. In such systems, stakes for individual peers would be higher as the cost of providing (and receiving) services would be very high (as opposed to almost zero cost associated with an audio file sharing, for example). It is thus reasonable to assume that all peers would behave selfishly in order to maximize the utility that they derive from the system.

Game theory [52] is an ideal tool to model a system with selfish nodes. We model the interaction among peers in a P2P system as an infinitely repeated game, and compute the Nash equilibrium strategies (i.e., the participation level) of nodes in such a game. Peers use game theory principles to determine when they should or should not serve others. We treat each peer as a rational, strategic player, who wants to maximize its utility by participating in the P2P system. Peers gain utility by obtaining services (resources) and loose utility (i.e., incur cost) while serving others. Since probability of obtaining service is dependent on one's reputation (which is gained only by serving others), peers strategize their actions such that their overall utility is maximized, i.e., they serve at a minimum level that maximizes their probability of obtaining service in future. Moreover, a system designer can use the game theoretic notion of *Nash Equilibrium* to analyze the strategic choices made by different peers, and study the overall efficiency of the system (including how it can improved).

When nodes also derive utility out of altruism, the probability of providing service would be higher than that given by the model presented in this chapter. Therefore, in some sense the model presented here provides a lower bound on the participation level for nodes below which they should not provide service.

6.2 Service Game

We assume the network lifetime to be infinitely long and divide the total time into individual time periods, represented by t for $t = 0, 1, \dots \infty$. In every time period each node gets a request for service. The other activity of each node during a time period is to obtain service for itself. Every service request maps to one or more service providers, which can be requested in parallel or sequentially. A request is assumed to be fulfilled when any of the requested service provider agrees to serve. For simplicity, we assume that a node always request exactly one service and is also requested exactly once for service in every time period. We assume a node's utility to be zero if it receives service more than once in a time period. Moreover, in an actual implementation, a node might receive multiple requests, however, some of those requests might be from low reputation clients and thus might be ignored. So the action {Serve} (as described below) corresponds to a situation when any one of the received request during a time period is served.

We model the interaction among peers as an infinitely repeated game. A game is played during each time period. In a game, denoted by G, nodes request service for themselves, and decide whether to serve others or not. Precisely, the game G is defined as follows.

Players: All the peers.

Actions: Each player's set of actions is {Serve, Don't serve}.[1]

Preferences: Each player's preferences are represented by the expected value of a payoff function that assigns value U when service is received and cost C when service is provided.

The *service game*, which is an infinitely repeated version of game G (i.e., when G is played over and over again in successive time periods) is represented by G^{∞}.

As stated earlier, the probability with which a player receives service is dependent on its current reputation. Reputation of player i in some time period t is denoted by R_t^i. Reputation of a node depends on its performance in the current time period as well as in prior time periods. Formally, we define R_t^i as follows.

$$R_t^i = R_{t-1}^i(1-\alpha) + \omega * \alpha, \quad 0 \leq \alpha \leq 1 \;\; t \geq 2 \tag{6.1}$$

In the above equation, ω is 1 when service is provided by player i in time period t, and is 0 otherwise. Therefore, we have $0 \leq R_t^i \leq 1$, i.e., the reputation of a player in every time period is always a value between 0 and 1 (including). Moreover, at $t = 0$ the reputation of all players is 0 and at $t = 1$ the reputation is given simply by ω. The parameter α is a constant and captures the importance assigned to

[1]The action corresponding to obtaining a service in a time period is not explicitly included in the model, as it is always assumed to take place.

the current performance of a player as opposed to its past performance for estimating its reputation. A high value of α means that more importance is assigned to a player's service in the current time period than its previous service record, and vice versa. Thus, when α is high, a node with even low reputation value can significantly improve its reputation by providing service in the current time period.

We assume that service information is readily propagated in the network and is available to all the players, i.e., each player when it serves another player propagates that information to as many other players as possible. The received service information is then recursively forwarded to other players. Therefore, it can be assumed that service information is propagated using a mechanism such as flooding, and is available to all the players.

6.3 Nash Equilibrium of the Service Game (G^{∞})

Now we evaluate the possible Nash equilibria of the service game (G^{∞}). By *Nash Folk Theorem* [52], if a^* is a Nash equilibrium action profile for some game G' then it is also the Nash equilibrium action profile when G' is played repeatedly infinite number of times.[2]

Therefore, finding the Nash equilibria of G^{∞} reduces to finding the Nash equilibria of game G. We evaluate both pure- and mixed-strategy Nash equilibrium of G. Since the proposed incentive mechanism based on a player's reputation links the benefit that a player draws from the system to its contribution - the benefit is a monotonically increasing function of a player's contribution. Thus, this is a non-cooperative game among the players, where each player wants to maximize its utility. The classical concept of Nash equilibrium points a way out of the endless cycle of speculation and counter-speculation as to what strategies the players should use, and is defined formally below (from [52]).

A Nash equilibrium is an action profile a^* with the property that no player i can do better by choosing an action different from a_i^*, given that every other player j adheres to a_j^*.

An equilibrium point is a locally optimum set of strategies (service probabilities, i.e., how much to serve others), where no player can improve its utility by deviating from the strategy.

[2]The notation a^* denotes the Nash equilibrium action profile of all the players, such that the Nash equilibrium action of player i in this equilibrium is given by a_i^*.

6.3.1 Pure Strategy Equilibrium

The action profile where all the players select the action {Don't serve} is a Nash equilibrium. This is because if any player i decides to serve, by selecting the action {Serve} instead, its payoff is $-C$, which is less than a payoff of 0 that it gets when it provides no service. The payoff of player i is $-C$ when it decides to serve because all the other players choose the action {Don't serve}, and therefore player i is unable to utilize its increased reputation to obtain service from others (and derive utility U in return).[3]

However, the action profile wherein all players choose the action {Don't serve} is an undesirable Nash equilibrium, since it means that no service is provided in the network. As a result the whole P2P system breaks down, and therefore, this equilibrium is an undesirable one. In light of this we argue that this action profile is an unlikely equilibrium and is not likely to be reached (especially when there is also altruism among network nodes to some extent).

The action profile where all the players select the action {Serve} is not a Nash equilibrium. This is easy to see because if everyone else is serving requests than the best strategy for any player is to deviate by switching its strategy to {Don't serve}. By doing so the player gets a payoff of U instead of $U - C$ when it also serves.

Thus, we conclude that the only pure strategy Nash equilibrium of G is when players select the action {Don't serve} (such that action {Don't serve} is selected in each time period in G^{∞}), which, however, does not appear to be a likely convergence state for any useful P2P system.

6.3.2 Mixed Strategy Equilibrium

We now consider the possibility of a mixed strategy Nash equilibrium of G, wherein players instead of deterministically selecting their actions randomize among their available set of actions. In other words, players select the action {Serve} in some time periods and the action {Don't serve} in others in G^{∞}.

We want to find a symmetric Nash equilibrium because all the players belong to the same population (i.e., assume the same role) and it is therefore easier (i.e., require no coordination among players)

[3]Similar result involving non-cooperation among players is predicted by the classical Prisonner's Dilemma game.

to achieve such an equilibrium. (A *symmetric Nash equilibrium* is an action profile a^*, which is a Nash equilibrium and $a_i^* = a_j^*$ for any two players *i* and *j*. Stated simply, in a symmetric Nash equilibrium all the players take the same action (deterministically or probabilistically)). If the players in a game either do not differ significantly or are not aware of any differences among themsleves, i.e., if they are drawn from a single homogeneous population, then it is difficult for them to coordinate, and a symmetric equilibrium, in which every player uses the same strategy, is more compelling. The argument of a single homogeneous population implies that all the peers in a P2P system have equivalent responsibilities and capabilities as everybody else.

Let there be such a mixed strategy Nash equilibrium in *G*, such that a player selects the action {Serve} with probability *p* and the action {Don't serve} with probability *1-p*. Here *p* is a non-zero value, i.e., both the actions are assigned positive probability by the mixed strategy of the player. Since the discussion here applies to all the players, therefore, for convenience we omit reference to a particular player and drop superscript *i* when defining reputation as given in Equation 6.1.

The expected payoff to a player in time period t when it selects the action {Serve} is $p(-C + R_t^{serve} * U)$. This payoff value we denote by $Payoff_{serve}$. Likewise, the expected payoff to a player in time period t when it selects the action {Don't serve} is $(1-p) * (R_t^{don't} * U)$. This payoff value we denote by $Payoff_{don't}$. A player's payoff when it provides service in a certain time period is $-C + R_t^{serve} * U$, which is the cost of providing service plus the utility it derives upon obtaining service. The term $R_t^{serve} * U$ captures the notion that the probability of obtaining service is directly proportional to one's reputation. Therefore, a player's expected payoff in a mixed strategy that selects the action {Serve} with probability *p* is given by $p(-C + R_t^{serve} * U)$. Likewise, one can obtain a player's expected payoff in a mixed strategy that selects action {Don't serve} with probability *1-p*. Here we have made an assumption that players first have an opportunity to serve others (and hence increase their reputation) before requesting service for themselves.

R_t^{serve} is a player's reputation when it provides service in time period t and $R_t^{don't}$ is a player's reputation when it does not provide service in time period t. From Equation 6.1, we obtain

$$R_t^{serve} = R_{t-1}(1-\alpha) + \alpha$$

and

$$R_t^{don't} = R_{t-1}(1-\alpha)$$

An important characterization of mixed-strategy Nash equilibrium of finite games (one where action set of players is finite) is the following (from [52]).

Each player's expected payoff in an equilibrium is its expected payoff to any of its actions that it uses with positive probability.

The above useful characterization of a mixed-strategy Nash equilibrium yields,

$$Payoff_{serve} = Payoff_{don't}$$

$$\Rightarrow p(-C + (R_{t-1}(1-\alpha)+\alpha) * U) = (1-p) * ((R_{t-1}(1-\alpha)) * U) \tag{6.2}$$

Solving Equation 6.2, we get

$$p = \frac{R_{t-1}U(1-\alpha)}{-C + 2R_{t-1}U(1-\alpha) + U\alpha} \tag{6.3}$$

It must be noted that the value p obtained above is not a constant, but varies in each time interval depending upon a node's reputation at the end of the previous time interval. The mixed-strategy $(p, 1-p)$ for actions {Serve, Don't serve}, respectively, is a mixed-strategy Nash equilibrium for the players. Assuming no collusion among nodes, if all the other nodes follow the above strategy, then the best strategy for any node is to also follow the above strategy (from the definition of Nash equilibrium). This is a symmetric mixed-strategy Nash equilibrium for G as well as G^{∞}. We argue that it is a more stable equilibrium than the one in which no service is ever provided by the nodes. This is because of the following two reasons. First, when no service is provided the network is not useful to any user. Second, in practice users, which derive finite utility from altruism, would always provide some service irrespective of how much they obtain in return. Therefore, it is unlikely to have a scenario in which no cooperation among nodes take place.

6.4 Properties of the Nash Equilibrium

In this section we study some interesting properties of the mixed-strategy Nash equilibrium derived above.

6.4.1 Simplicity of Calculating the Equilibrium Strategy

In the previous section we calculated the probability based on which nodes decide whether it is optimal for them to serve or not to serve. In each play of the game (or time period), players based on their reputation at the end of the prior time period decide whether they should provide service in the current time period or not. This probability as one can see does not remain constant from one period to another, and depends on a player's reputation at the end of the last time period. Players can calculate their reputation using Equation 6.1, since they know precisely their actions at each play of the game. Thus, determining the Nash equilibrium strategy is fairly straightforward for a player. It must be noted that there is an inherent assumption that peers get serviced based on their current reputation. The exact mechanism as to how that gets achieved (or is enforced) is outside the scope of the research here.

Figure 6.1 gives an example of how a peer's reputation might change over time by following the equilibrium strategy proposed above. In the figure, an increase in the reputation value correspond to time intervals when service is provided by the peer, and vice versa. As can be seen, the equilibrium strategy of players provides that over time their behavior, in terms of providing service to others, is similar to each other (i.e., independent of the initial state or reputation value).

6.4.2 Address the Problem of Free-Riding

Since the mixed-strategy Nash equilibrium assigns positive probability to the action {Serve} (i.e., players serve others with a positive probability in each time period), the problem of free-riding is minimized in the network. The simple game theoretic model presented here, wherein reputation is used as a basis for providing service, predicts that it is in every peer's (including the free-riders') best interest to serve others. Our simulations support this behavior as we found that the total service received by a node is largely balanced by the total service that it has to offer to others, as shown in Figure 6.2.

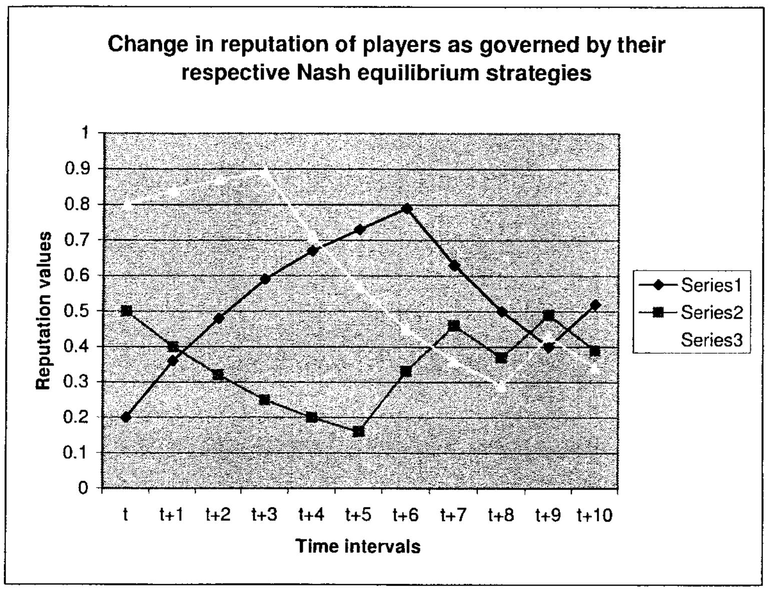


Figure 6.1 The figure shows the change in reputation values of players over time starting at time period t. The three players are assigned arbitrary reputation values to begin with and the results are shown for α equal to 0.5. Also, we set $U/C = 100$.

reputation \ alpha	0.2	0.4	0.6	0.8
0.2	14, 14	4, 4	2, 3	2, 2
0.4	16, 17	3, 4	1, 1	0, 1
0.6	12, 12	4, 5	2, 2	1, 1
0.8	17, 18	4, 5	2, 2	1, 1

Figure 6.2 The figure shows the total instances of service provided and received by a node over a period of 20 time intervals (first and second of the pair of numbers in each box represent the total instances of service provided and received, respectively). Irrespective of the initial reputation of a node and the value of α, the service received by a node is almost completely balanced by the service that it has to provide to others. Again, we set $U/C = 100$.

6.4.3 50 Percent Rule

An important property of the equilibrium emerges from Equation 6.3 that predicts the probability with which one should serve others. If we set $C << U$ (i.e., C can be ignored in Equation 6.3), then we have $p < 0.5$. In other words, if cost of providing service is negligible, Nash equilibrium of the service game predicts that players should serve each other less than 50 percent of the times when requested for service. This, although it appears to be very restrictive, is a consequence of the fact that all peers are selfish and are better off free-riding than serving others. Intuitively, if a peer knows that everyone else behaves selfishly, i.e., provide as little service as possible, then the best strategy for the peer cannot be to serve others most of the time (i.e., with probability greater than 0.5).

In terms of Nash equilibrium, the above result can easily be understood by the following simple counter-example. If all players are known to service requests most of the time then any player can easily increase its payoff by switching to a strategy where it free-ride on others, i.e., serves with probability less than 0.5. Thus, an action profile where peers generously serve each other cannot constitute a Nash equilibrium.

We believe that the above result is an important outcome of our game theoretic model of nodes' interaction in a P2P system, where all participants behave selfishly. Although, the above result is intuitively appealing, our model provides a proof based on game theory that explains such a behavior of peers.

6.4.4 Fairness - Equal Sharing of Cost of System Inefficiency

From our discussion in the previous subsection, where we concluded that serving with probability less than 0.5 is an optimal strategy (when $C << U$), one can see that the overall system efficiency is severely reduced. This is because at least half of the service requests in the system are not fulfilled.

However, on the positive side, the equilibrium strategy provides fairness in the sense that the cost of system inefficiency is not borne by a single peer, but is shared (in inverse proportion to one's reputation) among all peers. This is because each peer's request is likely to be turned down by the serving peer. We assume that if a peer's request at one peer is turned down it re-tries at some other candidate peer capable of serving the request. On average, the probability that a peer's request is successfully served in a time period is proportional to its current reputation.

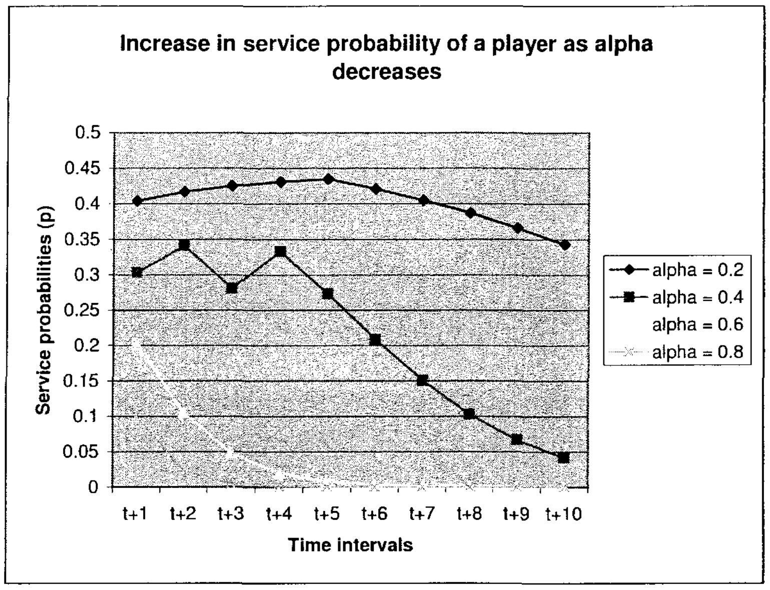


Figure 6.3 A decrease in value of α require players to serve each other with greater probability. An initial reputation value of 0.5 is used in all the cases for different values of α. Also, we set $U/C = 100$.

6.4.5 Decreasing α for Higher Contribution

As can be seen from Figure 6.3, a lower value of α shifts the service probability curve upwards. In other words, when α is low peers serve each other with higher probability. This is to be expected, since α determines how much importance is given to a peer's current performance as compared to its past service record. A low value of α (i.e., giving more importance to nodes past actions up to the current time period) means that peers need to continually provide service to be able to maintain high reputation and access service from the system. On the other hand, if α is high peers can easily increase their reputation in any period in which they provide service, irrespective of how cooperative they have been in past with regards to providing service to others.

Thus, a simple way to improve the system efficiency is to set α as low as possible. As shown in Figure 6.3, as we decrease α the value of p tends to 0.5 (which is the maximum possible service probability in a system with selfish peers when $C << U$; the same result can be obtained directly, if we set $C = 0$ and $\alpha = 0$ in Equation 6.3).

6.5 Related Research

Almost all the current research in P2P systems to overcome the free-riding problem relies on one of the following two incentive approaches - reputation [71, 72, 44, 45, 68, 69, 70] and monetary schemes [28, 79]. Most of the reputation based schemes assume that there are only a small set of malicious peers that do not provide service. It is assumed that most of the nodes are good and they would monitor the contribution of others to ensure that free-riding does not take place.

In a P2P system, a model where all the peers behave selfishly, however, appears to be more appropriate. This is true as P2P systems get used for more sophisticated application, such as distributed/grid computing, rather than simply sharing of music files, for example. In such scenarios, users have incentive to behave selfishly as stakes are typically higher. Recently game theory has been used to model the behavior of nodes in P2P systems. Below we describe two such recent papers. The first paper uses reputation and the second paper uses money as a form of incentive to motivate peers to cooperate.

6.5.1 [Chiranjeeb et. al.]

In [83] the authors use game theory to study the interaction of strategic and rational peers, and propose a differential service based incentive scheme to improve the system's performance. The authors first consider a simplified setting of homogeneous peers, where all peers derive equal benefit from everybody else. In this case it is shown that there are exactly two Nash equilibria, and there are closed form analytic formulae for these equilibria. The stability properties of these equilibria are investigated and it is shown that in a repeated game setting, the equilibrium with the better system welfare is realized. The authors use the symmetry of a homogeneous system to reduce the interaction of peers as a two-player game. This game is modeled as a Cournot-duopoly and the Nash equilibrium contribution of the players is calculated. The result is then extended to a N-player game.

For a system with heterogeneous peers, it was concluded that no closed form solution is possible and so simulations are used to study such a setting. The main findings are that the qualitative properties of the Nash equilibrium are impervious to (1) exact form of the probability function used to implement differential service, (2) perturbations like users leaving and joining the system, (3) non-strategic or non-rational players, who do not play according to the rules.

We believe that a major drawback of the proposed differential service mechanism is the difficulty of its implementation. This is because of the following two reasons - (1) It is assumed that a peer wishing to join the system first determines the benefit that it can derive from the system. If the benefit is larger than a critical benefit, then the peer's best option is to join the system and operate at the Nash equilibrium value of contribution. If on the other hand the benefit is less than the critical value, the peer is better off not joining the system. The benefit for a peer is the amount of resources contributed by other peers as well as the utility of those resources to the peer. Obtaining such information beforehand can be difficult and the issue is not adequately addressed in the paper. (2) Every request from a peer contains a metadata describing the contribution of the peer to the system. The differential service received by the peer is dependent on this information. Since peers have incentive to manipulate this information, the authors propose to use a *neighbor audit scheme*, in which peers continually monitor the contributions of their neighbors. We feel that such altruism on part of the peers is not reasonable to assume, especially when the underlying game model is uncooperative, and peers behave selfishly.

6.5.2 [Philippe et. al.]

The authors of [84] examine the design implications of the assumption that users selfishly act to maximize their own rewards. The authors construct a game theoretic model of the system and analyze equilibria of user strategies under several payment mechanisms. The idea is to encourage users to balance what they take from the system with what they contribute to the system. This is done by charging users for every download and rewarding them for every upload. The payment mechanisms differ primarily in the way that this amount of money is determined.

A micro-payment mechanism is described in which a centralized server at the end of each time period charges an agent the amount of money proportional to the difference between the number of downloads and uploads. Since users do not like micro-payments (having to decide before each download if a file is worth a few cents imposes mental decision costs), two variations of the basic scheme are also proposed. First is a quantized micro-payment mechanism in which users pay for downloads in blocks of b files, where b is a fixed parameter. At the end of a time period, the number of files downloaded by a user is rounded up to the next multiple of b, and the user is charged for these many blocks. Second is a point-based mechanism, which uses an internal currency called "points" instead of micro-payments. Like quantized micro-payments, this mechanism lets users trade a fixed amount of dollars for a block of b points. Even though files are paid for with points on a per-file basis, where dollars are concerned, the mechanism is essentially quantized.

The proposed micro-payment mechanism relies on a centralized entity (like an index server in Napster [9]) to keep track of all the file transactions in the network and accordingly charge the agents or nodes. In a completely distributed P2P network how the above payment mechanisms can be implemented is not clear. Also, the authors consider only three levels of user strategies for download (uploads) - no downloads (no uploads), moderate downloads (moderate uploads), and heavy downloads (heavy uploads). It is concluded that both heavy downloads and uploads constitute a unique Nash equilibrium strategy for the agents. But how the users should behave exactly within each level of strategy, for both downloads and uploads, is not clear.

6.6 Summary

In this chapter, we have proposed a simple mechanism based on nodes' reputation to minimize the free-riding problem prevalent in P2P systems. The mechanism addresses the free-riding problem as it predicts that even for selfish users serving others is the best strategy. Game theory is used to predict the optimum (Nash equilibrium) strategies of selfish nodes such that their profits are maximized. Game theory is also used to provide valuable insight into the behavior of individual nodes, as well as the performance of the overall system. Interestingly, game theory provide us a proof for some of the intuitive results, such as the strategy of serving less than 50 percent of the times when everybody else behaves selfishly.

The proposed game theoretic solution of the free-riding problem has several significant advantages - fairness, simple implementation, and ease of calculating optimum strategies.

CHAPTER 7. PROVIDING COMPLETE USER ANONYMITY

7.1 Overview

The goal here is to provide complete anonymity to the users of a P2P system, i.e., if a user initiates a request, say for data download or remote task execution, it should remain anonymous; likewise a user serving the request should not be traceable. For instance, in a commercial application, enterprises may be accessing resources residing at third-party computing facilities, e.g. delegating a computation-intensive job or acquiring a large amount of data. Since the third party could have significant findings about an enterprise activity if the enterprise identity is disclosed, anonymous communication in such scenarios is as important as other security issues.

Anonymity refers to the state that an entity is not identified in the communications with others. As discussed in [85], anonymous communication may have one or more of the following properties - sender anonymity, receiver anonymity, and unlinkability. Sender anonymity means that when a message is observed, the sender cannot be identified; receiver anonymity means that the receiver cannot be identified. Unlinkability means that the relationship between the sender and the receiver in the communication cannot be identified, even if sender anonymity or receiver anonymity cannot be guaranteed. Some anonymity mechanism may provide anonymity against one type of threat but not against another type. For example, using a proxy between senders and receivers may provide sender anonymity against the receiver and vice versa, but may not provide any anonymity against an eavesdropper who can observe all messages from and to the proxy.

In this chapter, we present a novel protocol for providing both client and server anonymity (and hence also unlinkability) in P2P networks. Here clients and servers are nodes in a P2P system. A node initiating a request for a service (or resource) is referred to as the client, and a node that serves the request is referred to as the server (for that particular request). The protocol assumes individual nodes or users to be utility maximizing agents, and relies on an auction mechanism for trading of resources among them. The resources here can refer to data files, storage capacity, or computation power (i.e., CPU cycles), etc. The protocol is light-weight, and incentive-compatible. Incentive compatibility implies that the protocol takes into account the selfishness of users; as we would see the utilities of users are maximized by truthfully following the protocol steps. The authors in [86] also propose the idea of using economic incentives for building decentralized anonymity infrastructure and motivating users to participate in such a system. Moreover, unlike other schemes, our proposed protocol does not rely on any trusted centralized entity or require specialized encryptions to be performed by the users. Thus, the protocol incurs very low overhead on the system and is light-weight.

We believe that the proposed scheme is easily deployable in a large untrusted Internet-scale setting. To the best of our knowledge, this is the first work that uses an incentive strategy to address the problem of both client and server anonymity in a single unified protocol.

7.2 Related Research

Anonymity for communication systems has been extensively studied, both for client-server and P2P computing models. Most of the existing anonymity protocols are for client-server computing model and hide the identity of the initiator (client). Initiator anonymity is provided using either rerouting based systems or non rerouting based systems. Crowds [87], Mix [91] and Onion-routing [94] employ rerouting based techniques to achieve initiator anonymity. In Crowds, anonymity on the World-Wide-Web (WWW) is provided by grouping the users into large and diverse groups, so that Web-servers are not able to learn the true source of the request. The user or the initiator submits a request which is forwarded to a random member of the crowd, which then forwards the request to the end server. In Mix and Onion-routing, the sender determines the rerouting path and encrypts the route in a layered fashion so that each intermediate node only knows its previous and next hop node.

Schemes which use a single hop intermediate rerouting path include Anonymizer [96] and Lucent Personalized Web Assistant (LPWA) [95]. An example of a non-rerouting based anonymous communication system is DC-Net (Dining Cryptographer's network) [97]. In this case a broadcast medium is used to achieve initiator and responder anonymity, and therefore it suffers from scalability issues.

In P2P publisher-subscriber systems, Freenet [88], Publius [98], FreeHaven [89] are examples of systems which provide publisher anonymity. Freenet is an adaptive P2P network application that permits publication and subscription of data in an anonymous fashion. Publius and FreeHaven use similar strategies for achieving publisher anonymity. While Publius splits the symmetric key used to encrypt and decrypt a document into n shares, FreeHaven splits the document into n shares. Any k of the n peers must be available to reproduce the key (in the first case) and the document (in the second case). While Gnutella [1] anonymizes only queries, GNUnet [99] provides anonymity for both queries and data transfers.

The authors in [90] suggest using a trusted index server for generating the rerouting path. Moreover, there is a substantial amount of overhead involved in encryptions and decryptions. Furthermore, only client anonymity is provided by this protocol. Other existing work [103, 100, 101] that study anonymity in Chord also focus only on client-side anonymity and not server-side anonymity.

The P5 [92] protocol uses a broadcast hierarchy to achieve both sender and receiver anonymity for peer to peer communications. The protocol uses public key cryptography with per-hop encryption and redundant noise packets to achieve a high degree of anonymity, thereby incurring a substantial overhead.

7.2.1 Our Approach

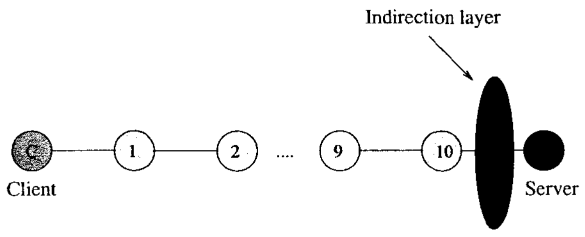


Figure 7.1 Figure depicting the intuition behind our anonymity providing strategy.

There are two essential building blocks that we exploit in our approach to simultaneously provide both client and server anonymity.

1. A peer-to-peer overlay topology and its associated routing mechanism that utilize intermediate nodes for routing a request (or response) from a client to server (or server to client).

2. An incentive scheme that makes it an optimal strategy for the intermediate nodes to keep information about their neighboring hops secret.

Our basic approach is illustrated using an example given in Figure 7.1, where a request from client C to server S is routed through 10 different intermediate nodes. In this figure, assuming that the above mentioned two building blocks work correctly, none of the nodes from 2 to 9 can know anything about the client or server. Moreover, node 1 upon receiving a request from C, cannot conclude if C is the actual client or just another intermediate node that itself received the request from someone else. (Note that since nodes exist in a virtual overlay topology, where each logical link might translate to several physical links, link sniffing by node 1 to determine whether C originated the request is also very difficult). Furthermore, an indirection layer introduced between nodes 10 and S creates an anonymous communication channel, which shields the identity of S from node 10. The functionality of the indirection layer is achieved in a completely distributed manner without relying on any trusted or centralized entity.

The design of the indirection layer and that of an appropriate incentive scheme are explained in the subsequent sections.

7.3 Model Assumptions and Limitations

We assume a Chord-based P2P network substrate that is used for network connectivity and resource lookups. We assume that a node's Chord identity (or simply Chord ID) is derived by applying an appropriate hash function to its IP address, and thus remains fixed for the node.

Some nodes in the network can be malicious, whose goal is to compromise the identity of the nodes that request and/or provide a service. Malicious nodes do not aim to maximize their profits (and in fact are prepared to incur loss), and can work in collusion, so as to identify who originated a request and who served it. We assume an *internal, passive*, and *static adversary* model [102], where - a) the adversary can observe lookup requests at compromised nodes but cannot observe the links (internal), b) the adversary can only observe the lookup requests but cannot modify them (passive) and c) the adversary chooses the nodes to compromise before the protocol starts (static). Assumptions (a) and (b) are generally true because the links under consideration are not actual physical links that can be monitored.

For a request initiated by a client, say *C*, for resource *R*, a network can be modelled as comprising of three types of entities - the client itself, the intermediate nodes (which forward the lookup message), and the server(s) capable of serving the request. We assume that a request for resource *R* can be met by any one of l servers, denoted by $S_{R_1}, S_{R_2}, \ldots, S_{R_l}$. (Here *R* denotes a name or ID that identifies the resource). We consider the process of looking up for and obtaining resource *R* as a one-shot game.

We propose a light-weight auction-based protocol, called the *anonymous lookup protocol*, for providing complete anonymity in peer-to-peer networks. For demonstrating the working of the anonymous lookup protocol, we consider an example of the lookup process initiated by the client *C* for resource *R*, as described previously. Specifically the protocol provides that the client *C* and any of the servers $S_{R_1}, S_{R_2}, \ldots, S_{R_l}$ can access and provide resource *R*, respectively, without their anonymity being compromised.

7.4 Details of the Anonymous Lookup Protocol

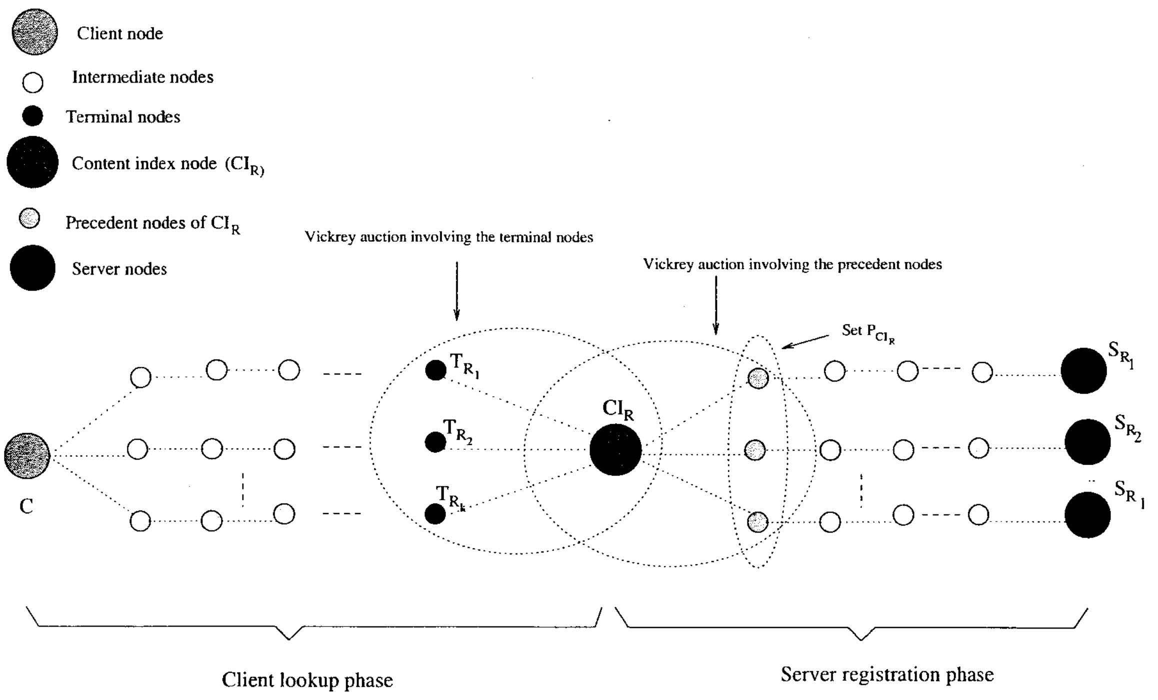

Client node
Intermediate nodes
Terminal nodes
Content index node (CI_R)
Precedent nodes of CI_R
Server nodes
Vickrey auction involving the terminal nodes
Vickrey auction involving the precedent nodes
Set P_{CI_R}
T_{R_1}
T_{R_2}
T_{R_k}
CI_R
C
S_{R_1}
S_{R_2}
S_{R_l}
Client lookup phase
Server registration phase

Figure 7.2 Figure depicting the operation of the anonymous lookup protocol.

The anonymous lookup protocol consist of two phases, as depicted in Figure 7.2. The first phase is the *server registration phase* and the second phase is the *client lookup phase.* The server registration phase occurs before the client lookup phase. Below we describe the steps involved in each of these phases. However, before that it is helpful to provide some definitions.

Definition 4. *Terminal nodes: Terminal nodes are the Chord successors of the hash values of a resource name. A lookup message from a client is first routed to a terminal node, which then forwards it to the server. We assume that there are t terminal nodes for each resource. For resource R, its terminal nodes are denoted by* T_{R_i} $\forall i \in \{1, \ldots, t\}$ *(for simplicity, we assume that t is a power of 2). If resource R hashes to Chord ID RI, then the t terminal nodes are the Chord successors of the following Chord IDs.*

$$(RI + \frac{2^m}{t} * (i - 1))mod(2^m), \forall i \in \{1, \ldots, t\} \tag{7.1}$$

Uniformly locating the terminal nodes around the Chord ring, as given by Equation 7.1, ensure that the lookup paths to different terminal nodes are as node disjoint as possible.

The routing of a message from a client to a terminal node may go through other intermediate nodes. This list of intermediate nodes along with the terminal node is referred to as a *request chain.* For now, request chains to different terminal nodes of a resource are assumed to be node disjoint, as shown in Figure 7.2.

Definition 5. *Content index node: Content index nodes are the Chord successors of the hash values of the contents of a resource. (In practice, to reduce the cost of computing the hash function, one can use a digest of the contents instead). For resource R, its content index node is denoted by* CI_R. *The routing of a message from a server* S_{R_j}, *where* $j \in \{1, \ldots, l\}$ *to a content index node may go through other intermediate nodes. This list of intermediate nodes along with the server is referred to as a service chain. For simplicity, service chains are also assumed to be node disjoint, as shown in Figure 7.2.*

7.4.1 Server Registration Phase

Each of the servers, $S_{R_1}, S_{R_2}, \ldots, S_{R_l}$, calculate the hash value of the contents of *R*, i.e., *hash(Content(R))*. For example, if *R* is the name of a file then the input to the hash function is the file itself. The servers then use the Chord lookup protocol to send a registration message $Msg_{register}$ to the content index node CI_R. Intermediate nodes also store the IP address of the node, referred to as the *precedent* node, from which the message was received.

Each registration message $Msg_{register}$ contains the following information - *hash(Content(R))*, resource name or ID (*R*), marginal cost (MC_{total}).

hash(Content(R)) is used to route the registration message to CI_R, the Chord successor of this value.

The name *R* is used by the intermediate nodes and also CI_R to know about the resource for which registration is being done.

MC_{total} contains S_{R_j}'s marginal cost $MC_{S_{R_j}}$ of providing the resource. An intermediate node on receiving the registration message updates MC_{total} by adding its own marginal cost to the received value.

CI_R receive *l* such registration messages, and thus knows that resource *R* can be obtained through any of the nodes that sent the registration message. These nodes comprise the set of precedent nodes of CI_R, and are represented by P_{CI_R}.

CI_R then uses the resource name *R* and Equation 7.1 to locate the corresponding terminal nodes $T_{R_i} \forall i \in \{1, \ldots, t\}$. The terminal nodes are informed by CI_R that resource *R* can be accessed through it.

The nodes in P_{CI_R} do not include the MC_{total} value in the registration message they send to CI_R. Once a request for resource *R* is received by CI_R from the terminal nodes, CI_R holds a second price sealed-bid auction (also called the Vickrey auction) with all its precedent nodes as the bidders. CI_R obtains the resource from the precedent node that offers to provide the resource at the lowest cost. The service chain containing the lowest cost bidder is called the winning service chain *WSC*. It must be noted that MC_{total} represents the minimum price that must be paid by CI_R in order to obtain the resource from the corresponding service chain.

7.4.2 Client Lookup Phase

The client *C* before initiating the lookup process estimates its utility (U_R^C) of the resource *R* to calculate the maximum price that it can offer for the resource. *C* then sends a separate lookup message towards all the terminal nodes of resource *R*, such that at most one message is sent out for all the terminal nodes that require going through the same next hop neighbor - the terminal node selected is one which is closest to that neighbor. As a result, the number of terminal nodes that are contacted during a client lookup phase may be less than the total number of terminal nodes for a resource, and therefore, the number of request chains formed, denoted by *k*, are typically less than *t*. Thus, we have $k \leq t$.

Together the parallel lookup messages towards different terminal nodes constitute a single lookup process initiated by client *C* for resource *R*. Each lookup message Msg_{lookup} contains the following information - address of one of the *k* terminal nodes (T_{R_i}), resource ID (R), maximum price offered (P_C), marginal cost (MC_{total}), request ID $Reqid_{public}$).

$Reqid_{public}$ identifies the lookup process such that CI_R on receiving the resource requests know that they pertain to the same lookup process. Thus, the same value of $Reqid_{public}$ is included in all the lookup messages.

MC_{total} contains *C*'s marginal cost MC_C. An intermediate node upon receiving the lookup message updates MC_{total} by adding its own marginal cost to the received value.

An intermediate node on receiving a lookup message routes it to the next hop neighbor, and this process continues till the message reaches the desired terminal node, which in turn contacts CI_R to obtain the resource. CI_R receive *k* such requests and from the $Reqid_{public}$ values knows that all the requests pertain to the same lookup process. CI_R then holds a second price sealed-bid auction (also called the Vickrey auction) with all the terminal nodes as the bidders. CI_R provides the resource to the terminal node that offers the highest price. The request chain containing the highest bidder, i.e., the winning terminal node, is called the winning request chain *WRC*.

7.4.2.1 Using Vickrey Auction for Resource Trading

As explained above, CI_R holds two separate Vickrey auctions - one with the terminal nodes as the bidders, and the other with the nodes in P_{CI_R} as the bidders.

In Vickrey auction, the highest bidder wins the auction, but the price that it has to pay is equal to the second highest bid. Vickrey auction in its most basic form is designed to be used by altruistic auctioneers, which are concerned with overall system efficiency or social good as opposed to self-gains. Self-interested auctioneer is one of the main reasons why Vickrey auction did not find widespread popularity in human societies [65].

Since, CI_R (the auctioneer) behaves selfishly and tries to maximize its profit, the auction process involving the terminal nodes (precedent set nodes) needs to ensure the following.

- Selecting the highest (lowest) bidder is the best strategy for CI_R.
- The price paid by the highest (lowest) bidder is indeed equal to the second highest (lowest) bid, i.e., CI_R should reveal true second highest (lowest) bid to the highest (lowest) bidder.
- Collusion among CI_R and the bidders should not be possible.

In view of the above requirements, we use a two-phase Vickrey auction protocol, which was proposed and evaluated for its correctness and robustness in Chapter 4.

7.4.2.2 Secure Vickrey Auction to Determine the Winning Terminal Node

We now explain in detail the auction process involving the terminal nodes only. The auction process involving the nodes in P_{CI_R} is carried out using exactly the same procedure, except for the fact that the winner now is the one with the lowest bid, i.e., cost value.

We denote the highest and second highest bids by M_1 and M_2, respectively. The price offered by a terminal node to CI_R is equal to $P_C - MC_{total}$. (On the other hand, the bid offered by a node in P_{CI_R} is simply MC_{total}). The amount of profit made by the *WRC* is equal to $(M_1 - M_2)$. This profit is shared fairly among the nodes of the *WRC* (and the client) in proportion to their marginal costs, i.e., nodes with higher marginal costs get a higher proportion of the total profit, and vice versa.

CI_R employs a two-phase Vickrey auction to select the highest bidder and determine the price at which the resource is provided. In the first phase, the bidders send encrypted copies ($E(randKey_i; b_i)$) of their bids in message Msg_{bid} to CI_R. Here $E(randKey_i; b_i)$ is the encryption of bid value b_i of terminal node T_{R_i} using a randomly chosen secret key $randKey_i$. Each message Msg_{bid} also includes the $Reqid_{public}$ value received by the terminal nodes, so that CI_R can determine that the bids pertain to the same lookup process. The received encrypted bids are sent by CI_R back to all the bidders in message $Msg_{bid-reply}$. Since after receiving $Msg_{bid-reply}$, the bidders have encrypted copies of all the bids (total k such bids), CI_R is unable to (undetectedly) alter existing or add fake bids.

In the next and last phase of the auction, each bidder after receiving the message $Msg_{bid-reply}$, sends its secret key in message Msg_{key} to CI_R. The received key values are now sent by CI_R back to all the bidders in message $Msg_{key-reply}$. At the end of this phase, CI_R and all the bidders are able to open the encrypted bids and find out about the highest and second highest bids.

CI_R then sends a message Msg_{cert} to the winning terminal node certifying that it has won the auction. The received certificate is forwarded along the reverse lookup path until it reaches *C*. *C* then finds out that the resource has been looked up and is available at a price within its initial offer of P_C. Msg_{cert} contains the following information - highest bid M_1, second highest bid M_2, total marginal cost MC_{total}. (MC_{total} is received by CI_R in Msg_{bid}). The corresponding Msg_{cert} message in the auction involving the nodes in P_{CI_R} include only the information about the lowest and second lowest bid.

The information in messages Msg_{cert} and Msg_{lookup} ($Msg_{register}$) allow the intermediate nodes, including the winning terminal node, to calculate their reward for being part of the *WRC* (*WSC*). The knowledge of the auction results also enables *C* to determine the price that it finally has to pay for *R*. The calculation of the exact payoffs received by nodes are discussed in the next section.

At the end of the two auctions, the resource is obtained via the lowest cost precedent node (from the server on the *WSC*), and provided to the terminal node on the *WRC*. The terminal node sends the resource to the client along the reverse lookup path. (Since the nodes on *WSC* and *WRC* have to both receive and send the contents of resource *R*, we assume that the $Msg_{register}$ and Msg_{lookup} messages also include the size information of resource *R*. This enables the intermediate nodes to calculate their

$Msg_{register}$	$hash(Content(R)), R, MC_{total}$
Msg_{bid}	$E(randKey_i; b_i), R$
$Msg_{bid-reply}$	$\cup E(randKey_i; b_i), R$
Msg_{key}	$randKey_i, R$
$Msg_{key-reply}$	$\cup randKey_i, R$
Msg_{cert}	M_1', M_2', R

Table 7.1 Various messages comprising the server registration phase and Vickrey auction involving the nodes in P_{CI_R}. The resource name *R* is included in all the messages so that the receiver can correctly establish the context for the received message (for example it is possible for CI_R to be the content index node for some other resource also). M_1' and M_2' are the lowest and second lowest bids, respectively.

Msg_{lookup}	$T_{R_i}, R, P_C, MC_{total}, Reqid_{public}$
Msg_{bid}	$E(randKey_i; b_i), MC_{total}, Reqid_{public}$
$Msg_{bid-reply}$	$\cup E(randKey_i; b_i), Reqid_{public}$
Msg_{key}	$randKey_i, Reqid_{public}$
$Msg_{key-reply}$	$\cup randKey_i, Reqid_{public}$
Msg_{cert}	$M_1, M_2, Reqid_{public}, MC_{total}$

Table 7.2 Various messages comprising the client lookup phase and Vickrey auction involving the terminal nodes. The value $Reqid_{public}$ is included in all the messages so that the receiver can correctly establish the context for the received message.

marginal cost values for participating in the lookup transaction. Note that the client can only estimate the size of resource *R*).

Figure 7.3 summarizes the exact sequence of steps followed in the proposed anonymous lookup protocol. For an easy reference, the various messages used during the server registration and client lookup phases, along with the information they contain, are also summarized in Table 7.1 and Table 7.2, respectively.

7.4.3 Distributing Reward to Nodes in *WRC* and *WSC*

For any node in *WRC*, say *x*, its payoff Pay_x is calculated as follows.

$$Pay_x = MC_x + (\frac{MC_x}{MC_{total}} * (M_1 - M_2)) \tag{7.2}$$

```
/* SERVER REGISTRATION PHASE START */

Step 1: Server(s) with resource R register themselves by sending a registration message $Msg_{register}$ to the content index node, which is the Chord successor of hash(Content(R))
-  Intermediate nodes update the value of $MC_{total}$ before forwarding the registration message
-  Registration messages reach $CI_R$, which is the content index node for resource R. It must be noted that the registration messages received by $CI_R$ do not include in them the $MC_{total}$ values
/* $CI_R$ now knows that resource R can be obtained through its precedent nodes, which are represented by $P_{CI_R}$ */
Step 2: $CI_R$ uses the resource name R to locate the corresponding terminal nodes $T_{R_i} \forall i \in \{1,\dots,t\}$

Step 3: The terminal nodes are informed that resource R can be accessed through $CI_R$


/* SERVER REGISTRATION PHASE FINISHES */

/* CLIENT LOOKUP PHASE START */

Step 4: Client initiates the lookup process by sending a lookup message $Msg_{lookup}$ towards $T_{R_i} \forall i \in \{1,\dots,k\}$
-  Intermediate nodes update the value of $MC_{total}$ before forwarding the lookup message
-  Lookup messages reach the terminal nodes


/* VICKREY AUCTION STARTS (involving the terminal nodes) */
/* Phase I */
Step 5: Terminal nodes on receiving $Msg_{lookup}$ send $Msg_{bid}$ to $CI_R$

Step 6: $CI_R$ waits for k $Msg_{bid}$ messages (i.e., bids) or till some maximum time $\tau$
-  Bids are identified as belonging to the same lookup process by using the value $Reqid_{public}$

Step 7: Server sends message $Msg_{bid-reply}$ to the terminal nodes
-  After the above step the bidders have encrypted copies of all the bids

/* Phase II */
Step 8: Terminal nodes send their secret key to $CI_R$ in message $Msg_{key}$

Step 9: $CI_R$ replies with a message $Msg_{key-reply}$ distributing the secret keys among the bidders


/* VICKREY AUCTION FINISHES */

/* At the end of the above Vickrey auction, $CI_R$ knows the maximum price that it can offer for resource R. It then solicits bids from its precedent nodes. It must be noted that these bids correspond to the minimum price at which the precedent nodes can provide the resource */

/* VICKREY AUCTION STARTS (involving the nodes in $P_{CI_R}$) */

/* Phase I */
Step 10: Nodes in $P_{CI_R}$ send $Msg_{bid}$ to $CI_R$

Step 11: $CI_R$ waits for l $Msg_{bid}$ messages (i.e., bids) or till some maximum time $\tau$
-  Bids are identified as belonging to the same lookup process by using the resource name R

Step 12: $CI_R$ sends message $Msg_{bid-reply}$ to the bidders
-  After the above step the bidders have encrypted copies of all the bids

/* Phase II */
Step 13: Bidders send their secret key to $CI_R$ in message $Msg_{key}$

Step 14: $CI_R$ replies with a message $Msg_{key-reply}$ distributing the secret keys among the bidders
```

Step 15: CI_R sends message Msg_{cert} to the precedent node with the lowest bid value. The receiving node in turn propagates the message along the service chain until the message reaches the server

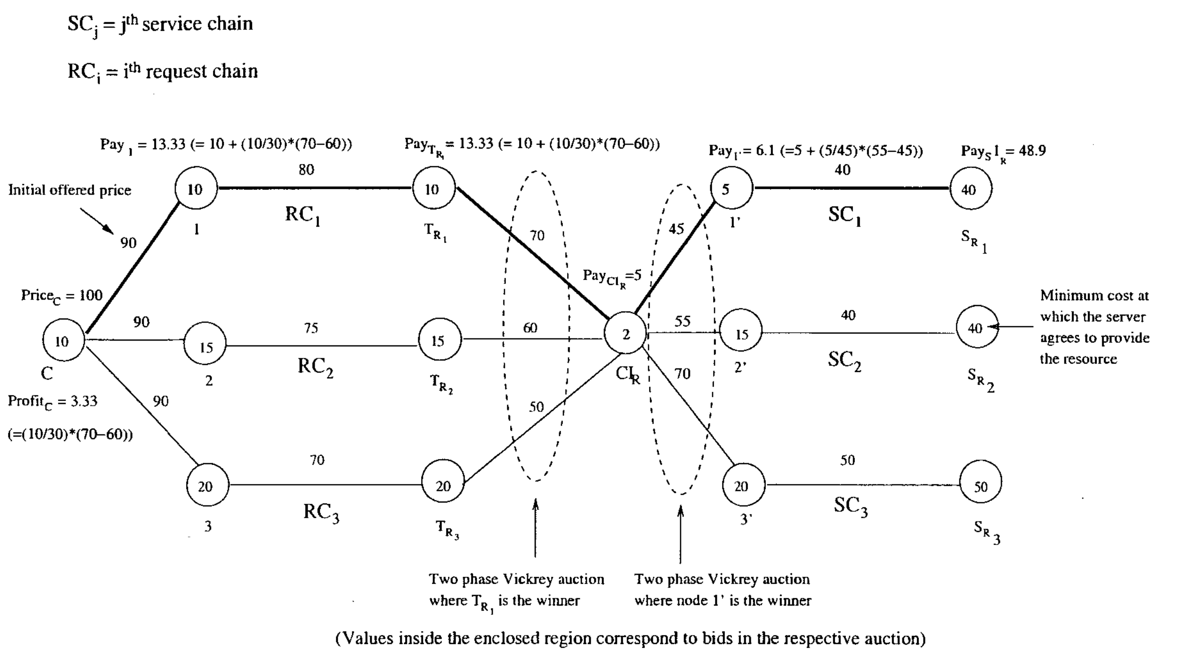
/* By using the contents of messages $Msg_{register}$ and Msg_{cert}, nodes along the *WSC* know the payoff they have to make to their precedent nodes */
Step 16: The server, which is part of the *WSC*, supplies the requested resource. The resource is again propagated along the *WSC* until it reaches CI_R

Step 17: CI_R then sends message Msg_{cert} and resource R to $T_{R_{WRC}}$. This message, along with the resource, is sent to *C* using the reverse lookup path

Step 18: *C* after keeping its profit share gives the remainder of its initial offer to the next hop node along the *WRC*. The next hop node then keeps its payoff amount and sends the remaining to its next hop, and so on. This process is repeated until CI_R receives a payoff of M_2 from $T_{R_{WRC}}$

Step 19: CI_R gives a payoff of M_2' to the winning precedent node, i.e., the one with the lowest cost of providing the resource. (We assume that $M_2 - M_2' > MC_{CI_R}$, thereby giving a net profit to CI_R). Each node along the *WSC* after keeping its payoff amount sends the remaining to its precedent node. This process is repeated till the server that is part of the *WSC* receives its payoff
/* CLIENT LOOKUP PHASE FINISHES */

Figure 7.3 Anonymous lookup protocol steps



SC_j = jth service chain
RC_i = ith request chain
Pay_1 = 13.33 (= 10 + (10/30)*(70−60))
$Pay_{T_{R_1}}$ = 13.33 (= 10 + (10/30)*(70−60))
$Pay_{1'}$ = 6.1 (=5 + (5/45)*(55−45))
$Pay_{S_{R_1}}$ = 48.9
Initial offered price
90
80
RC_1
40
SC_1
10
1
10
T_{R_1}
70
45
5
1'
40
S_{R_1}
$Price_C$ = 100
Pay_{CI_R}=5
Minimum cost at which the server agrees to provide the resource
90
10
C
15
2
75
RC_2
15
T_{R_2}
60
2
CI_R
55
15
2'
40
SC_2
40
S_{R_2}
$Profit_C$ = 3.33
(=(10/30)*(70−60))
90
70
50
70
20
3
RC_3
20
T_{R_3}
20
3'
50
SC_3
50
S_{R_3}
Two phase Vickrey auction where T_{R_1} is the winner
Two phase Vickrey auction where node 1' is the winner
(Values inside the enclosed region correspond to bids in the respective auction)

Figure 7.4 An example illustrating how the payoffs are distributed among the *WRC* and *WSC* nodes based on their marginal costs.

The profit share of *C*, i.e., the portion of its initial offer that it saves or gets to keep, is similarly calculated as given below.

$$Profit_C = (\frac{MC_C}{MC_{total}} * (M_1 - M_2)) \tag{7.3}$$

Likewise, let M_1' and M_2' be the lowest and second lowest bid, respectively, in the auction involving the nodes in P_{CI_R}. Then for any node in *WSC*, say *x*, its payoff Pay_x is calculated as follows.

$$Pay_x = MC_x + (\frac{MC_x}{M_1'} * (M_2' - M_1')) \tag{7.4}$$

The payoff received by CI_R is equal to,

$$Pay_{CI_R} = M_2 - M_2' \tag{7.5}$$

The example depicted in Figure 7.4 illustrates the payoffs received by different nodes in *WRC* and *WSC*. Both *WRC* and *WSC* are darkened in the figure. Numbers inside the nodes represent their marginal cost values. The payoffs to the nodes on *WRC* and *WSC* are also indicated in the figure. For example, payoff to node *1*, which is part of *WRC*, is 13.33 (=10 + (10/30)*(70-60)), and the payoff to node *1'*, which is part of *WSC* is 6.1. *C*'s profit share is 3.33 (= (10/30)*(70-60)). Thus, *C* effectively has to pay only 86.67(=100-10-3.33) for a resource whose utility to it (after deducting the marginal cost) is in fact 90. Therefore, the use of Vickrey auction ensures that everyone, including the client, server, terminal nodes, and intermediate nodes constituting the *WRC* and *WSC* benefit, i.e., earn more than their marginal costs of participating in the lookup process. This potential of earning higher profits motivate nodes to share their resources and forward messages for others.

7.5 Anonymity Analysis

In the anonymous lookup protocol we use the fact that nodes are selfish, and in order to maximize their payoffs they have incentive not to reveal information about their precedent nodes (which send the lookup or registration messages) to their next hop neighbors. This is because otherwise the precedent and next hop nodes can directly negotiate among themselves and by-pass the nodes in-between, and consequently there will be less nodes with whom the profit will have to be shared. Our incentive-based strategy of lookups in Chord allows us to exploit this inherent property of information hiding,

A:	Adversary
u:	Total number of adversaries (in case of more than one)
$d(ij)$:	Distance (number of identifiers) between nodes i and j
$d_{binary}(ij)$:	Binary representation of distance between nodes i and j
L:	The event that the adversary does not lie on any lookup chain
S(x):	Anonymity set for event x
d(S):	Degree of anonymity on set S

Figure 7.5 Notations for anonymity analysis

and anonymity is thus naturally provided by the proposed protocol. Also, note that at no point in the operation of the protocol, the identities of the client or server(s) are revealed - none of the messages contain this information. Even the next hop neighbors of C (S_{R_i}) do not know that the request was originated (served) at C (S_{R_i}). As can be seen the functionality of the indirection layer (as described in Section 7.2.1) is implemented by nodes between the terminal nodes and server nodes. The nodes constituting the indirection layer are configured during the server registration phase.

Unlike in the traditional Chord protocol (or any other DHT based system), where the successor nodes of keys (referred to as the terminal nodes in our case) either directly store the key value or the address of a node containing the key value, we require the terminal nodes to store the address of the content index node. This is important as we want to provide both client and server anonymity, and otherwise server identity is always known to the terminal nodes. One might argue that a single request- and service-chain might also be used (by using a single terminal node and registering the server(s) directly at that node) to provide both anonymity and resource trading. However, in such a scenario both the client and server(s) would have to speculate about the other's offer and also how much the intermediate nodes would charge for routing. Moreover, the intermediate nodes, in order to maximize their profits, would have to speculate about the cost value they should reveal while still ensuring that the offer received by the auctioneer (content index node) from the client side is more than the minimum price asked by the service chain. To avoid such speculations (and counter-speculations) and enable fast resource trading, we use Vickrey auction. Vickrey auction is used on both the client and server side to decide how the resource is eventually priced. In summary, the two layer indexing scheme, and using the content index node, allows us to carry out the Vickrey auction protocol, and separate the client side of the lookup process from that of the server side.

After the informal reasoning given above, we now formally investigate the anonymity provided by the proposed anonymous lookup protocol. Since the anonymous lookup protocol is symmetric on both client and server side and resembles an "hour-glass" model, below we analyze client anonymity only. Similar arguments would apply for proving server side anonymity as well. Also, to make the derived equations more tractable, we set *N* equal to 2^m, where *m* is the number of bits in a Chord ID. However, the plots given in Figures 7.7 and 7.8 are for much smaller values of *N*, and even these small values provide a high degree of anonymity. We use the following metrics - *Average Anonymity Set* and *Degree of Anonymity*, which are commonly used to evaluate the strengths of an anonymous system. Notations that are used in our analysis are summarized in Table 7.5.

Definition 6. *Average Anonymity Set: Anonymity set, represented by S, is defined as the set containing all possible initiators of a lookup request as perceived by the adversary set. The average (or expected) anonymity set is the expected value of* $|S|$.

Definition 7. *Degree of Anonymity:We use Entropy [104] to measure the degree of anonymity of a system. If X is a random variable representing the initiator of a lookup chain, then the entropy,* $H(X)$ *is a measure of the information content of the probability distribution of* X. *More formally,*

$$H(X) = -\sum_{x \in S} Pr(X = x) \log_2 Pr(X = x) \tag{7.6}$$

Using this definition of entropy, we calculate the degree of anonymity on S as,

$$\begin{aligned} deg(S) &= \frac{H(X)_{aposteriori}}{H(X)_{apriori}} \\ &= \frac{-\sum\limits_{x \in S} Pr(X = x) \log_2 Pr(X = x)}{\log_2(N-1)} \end{aligned} \tag{7.7}$$

The apriori entropy corresponds to the information that the adversary has before observing any lookup request, and therefore it can only exclude itself from the anonymity set. In the following subsections we consider the possible threat models that are relevant for the anonymous lookup protocol.

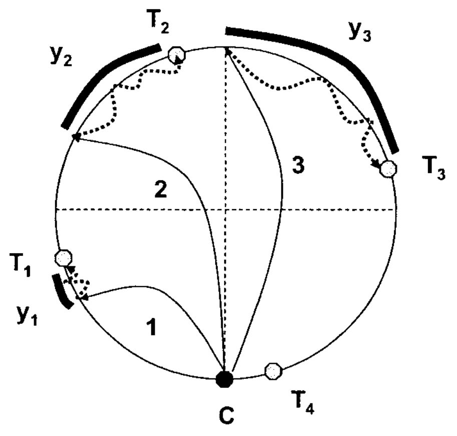


Figure 7.6 Multiple request chains initiated by *C*

7.5.1 Threat Model A (Single Adversary)

From the perspective of the adversary A, the anonymous lookup protocol provides a high degree of anonymity to the client *C*, as summarized by the following theorem.

Theorem 5. *The average size of the anonymity set for C is at least* $2^m - m(1 - 0.5^m)$

Proof. The average size of the anonymity set is given by,

$$E(|S|) \quad = \quad Pr(L).|S(L)| + Pr(\overline{L}).|S(\overline{L})| \tag{7.8}$$

Now to derive $Pr(\overline{L})$ we use the following lemma.

Lemma 6. *For the lookup initiated by C, A lies on at most a single request chain.*

Proof. Refer to Appendix 8. □

From Lemma 6, we can say that request chains are node disjoint, i.e., at most a single request chain passes through A. Therefore, A will be on a request chain iff it lies on one of the y_i regions (as indicated in Figure 7.6). The region y_i is the Chord distance along the clockwise direction between the i^{th} finger of *C* and the terminal node (when there is a terminal node between the i^{th} and $(i+1)^{th}$ fingers), which is closest to that finger. From the property of Chord, the number of hops in the region y_i are $\log(y_i)$. So the probability that A lies on one of these regions (i.e., y_is) is given by,

$$Pr(\overline{L}) = \frac{\log(\sum_{i=1}^{i=k} y_i)}{N} = \frac{\log(\sum_{i=1}^{i=k} y_i)}{2^m}$$

For the worst case (best case for the adversary), we can say that

$$\frac{\log(\sum_{i=1}^{i=k} y_i)}{2^m} \leq \frac{\log(\sum_{i=1}^{i=m} y_i)}{2^m} = \frac{\log(N)}{2^m} = \frac{\log(2^m)}{2^m} = \frac{m}{2^m} \tag{7.9}$$

Now assuming that the adversary lies on one of the request chains and uses its q^{th} finger for routing the request, we get the following expression for the average size of the anonymity set.

$$\begin{aligned} E(|S|) &= (1 - \frac{m}{2^m})|S(L)| + \frac{m}{2^m}|S(\overline{L})| \\ &= (1 - \frac{m}{2^m})(N-1) + \frac{m}{2^m}2^{m-q} \end{aligned} \tag{7.10}$$

The average size of the anonymity set is a function of q, and the minimum value of $|E(S)|$ is obtained when $q = m$ (adversary uses its m^{th} finger to route the lookup). Substituting in 7.14, we get the following lower bound.

$$E(|S|) \geq 2^m - m(1 - 0.5^m) \tag{7.11}$$

□

The auctioneer (CI_R) acting as an adversary is a specific case of this threat model, and has similar analysis for the average anonymity set size. The auctioneer only know the identity of the terminal nodes through which it receive the lookup requests.

7.5.2 Threat Model B (Multiple Adversaries)

It is possible to have multiple adversaries in a network that can collude, i.e., share their information, in order to determine from where the request originated. Let there be u number of such adversaries. The robustness of the proposed protocol against the multiple adversary scenario is demonstrated by the following lemma.

Lemma 7. *Adversaries lying on all the request chains cannot collude to further reduce the size of the anonymity set than that available with the most downstream adversary.*

Proof. Consider a scenario in which the number of adversaries in the system is so large that there is an adversary on all the k request chains. We need only consider the most downstream adversary in each of the request chains, since they are closest to the client. (In a Chord ring, a node has more information about the region of the identifier space that is closer to it than about a far away region). Let $A_1, A_2, \ldots, A_k$ be k such adversaries lying on request chains $1, 2, \ldots, k$, respectively. The corresponding anonymity sets and finger table entries used by these adversaries are represented by $S_1, S_2, \ldots, S_k$ and $q_1, q_2, \ldots, q_k$, respectively. Without loss of generality, let $q_1 > q_2 > \ldots > q_k$.

From the property of the Chord routing protocol and also as given in [103], we have that q_1 least significant bits of the adversary A_i are the same as the q_1 least significant bits of the client C. Therefore, we will have $|S_1| = 2^{m-q_1}$. Similarly, for any j, where $2 <= j <= k$, we have that q_j least significant bits of the adversary A_j are the same as the q_j least significant bits of the client C, and hence $|S_j| = 2^{m-q_j}$. But since $q_1 > q_2 > \ldots > q_k$, these q_j bits would be a suffix of the least significant q_1 bits of A_1.

Now it is easy to see that, $S_1 \bigcap S_2 \bigcap \ldots \bigcap S_k = S_1$. In other words, the most downstream adversary cannot further reduce its anonymity set size by using the information available with the adversaries on other chains. $\square$

Based on the above observation we now calculate the expected size of the anonymity set for multiple adversaries.

Theorem 6. *The average size of the anonymity set is at least* $2^m - um(1 - 0.5^m)$

Proof. Again we have,

$$E(|S|) = Pr(L).|S(L)| + Pr(\overline{L}).|S(\overline{L})| \tag{7.12}$$

For a large value of N,

$$Pr(L) = (1 - \frac{m}{2^m})^u \tag{7.13}$$

$\square$

Using the approximation $(1-x)^u \approx 1-ux$ we have the following expression for the average size of the anonymity set.

$$E(|S|) \;=\; (1-\frac{um}{2^m})(N-1)+\frac{um}{2^m}2^{m-q} \tag{7.14}$$

From Lemma 7, the anonymity set of the most downstream adversary (closest) to the client is contained (subset) in the anonymity sets of all the other adversaries. Therefore, q in the above equation is the finger used by the most downstream adversary to route a lookup. Substituting $q=m$, we get a lower bound on the average size of the anonymity set.

$$E(|S|) \;\geq\; 2^m - um(1-0.5^m) \tag{7.15}$$

7.5.3 Threat Model C (Multiple Adversaries Populate the Finger Table of C)

We now consider a special threat model in which multiple adversaries are the first-hop nodes of *C*. Since from Lemma 6 we know that the request chains are disjoint, two or more first-hop adversaries receiving a lookup request can accurately identify the client *C*. However, the probability of such an event happening is very low, as shown below.

Lemma 8. *The probability of two or more adversaries being the first hop nodes of C is very small.*

Proof. From Lemma 6, we know that the request chains are node disjoint. Therefore, if there are *k* request chains, then there are exactly *k* first-hop possible positions that the adversaries can occupy. Let *X* be the event that two or more adversaries occupy these *k* positions, *Y* be the event that no adversary lies on these *k* first-hop positions, and *Z* be the event that exactly one adversary lies on one of these *k* first-hop positions.

Then,

$$Pr(X) \;=\; 1-Pr(Y)-Pr(Z) \tag{7.16}$$

For a large value of *N*,

$$Pr(Y) \approx (\frac{N-k}{N})^u \tag{7.17}$$

$$Pr(Z) = \binom{u}{1}(\frac{k}{N})(\frac{N-k}{N})^{u-1} \tag{7.18}$$

Therefore,

$$\begin{aligned} Pr(X) &\approx 1 - (\frac{N-k}{N})^u - \binom{u}{1}(\frac{k}{N})(\frac{N-k}{N})^{u-1} \\ &\approx 1 - (1 - \frac{ku}{N}) - \frac{ku}{N}(1 - \frac{(u-1)k}{N}) \\ &\approx \frac{u^2k^2}{N^2} \end{aligned} \tag{7.19}$$

For large networks, we typically have ($N >> ku$) and hence $Pr(X)$ is very small. (The maximum possible value of k is only m). □

7.5.4 Degree of Anonymity Calculation

Based on the average anonymity set size values calculated in the previous sections, we now give a generalized expression for the degree of anonymity for the proposed anonymous lookup protocol.

$$deg(S) = \frac{H(X)_{aposteriori}}{H(X)_{apriori}} \approx \frac{\log_2(2^m - um(1 - 0.5^m))}{\log_2(2^m)} \tag{7.20}$$

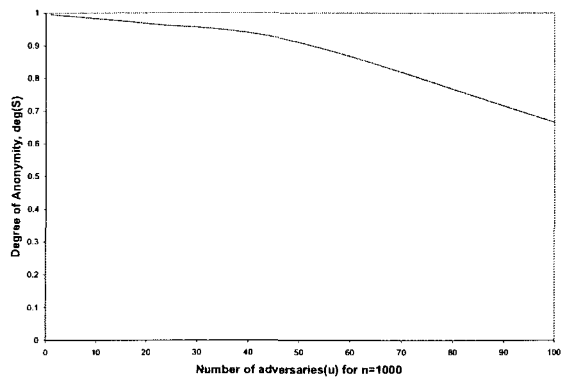


Figure 7.7 Variation of degree of anonymity with size of adversary set when $N = 1000$.

The plots of Equation 7.20 show that a very high degree of anonymity is achieved even when a significant fraction of the nodes are controlled by adversaries. Figures 7.7 and 7.8 show the variation of degree of anonymity with the number of adversaries present in a network of size 1000 and 50000,

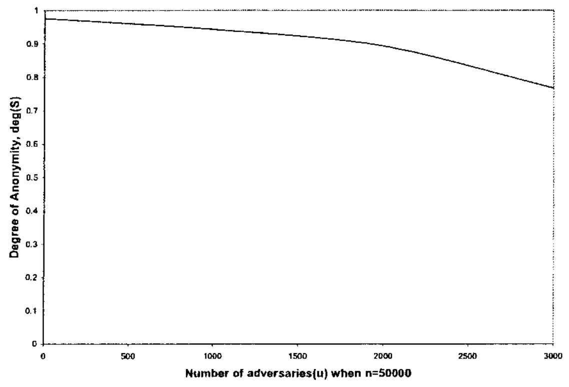


Figure 7.8 Variation of degree of anonymity with size of adversary set when $N = 50000$.

respectively. For the case when $N = 50000$, $deg(S)$ is as high as 0.8 even when 6% of the nodes are malicious. In information theoretic terms, this means that about 80% of the bits of the client's identity remains hidden from the adversaries. Moreover, our analytical expression for the degree of anonymity is independent of the number of request chains and the number of terminal nodes, with the implication that irrespective of the number of request chains initiated by a client, a very high degree of anonymity is achieved.

7.6 Protocol Overhead

We must admit that the proposed protocol incurs some extra over overhead, which is mainly due to the following two reasons.

1. Message communication involved in formation of request chains and service chains.

2. Use of monetary transactions among nodes.

3. Sending of data using multiple hops from the selected server to the client.

4. Computations involved in message encryption and decryption to achieve message non-repudiation

The maximum message processing overhead is incurred by CI_R, which is $O(k + l)$. The message overhead of the client is $O(\log k)$. The number of messages processed by an intermediate node and a server are $O(1)$. The maximum number of nodes involved in the lookup process are $O((k+l) * \log N)$, where k and l are the number of request chains and service chains, respectively, and $O(\log N)$ is the length of each chain. Thus, we can see that the protocol incurs a reasonable overall message overhead.

Several researchers have built light-weight payment mechanisms [28, 27, 26] for P2P systems. These mechanisms can easily be integrated with the anonymous lookup protocol. Also, in Chapter 4, we explained how reputation can be used as a substitute for reward, thus obviating the need for an expensive electronic infrastructure for money payments. Therefore, we believe that the requirement of monetary transactions by our protocol should not impose a significant overhead on the system.

Moreover, we are exploring ways for enhancing the protocol to minimize the number of hops required to send data from server to client. The strategy will be to send data separately, rather than sending it with Msg_{cert}.

7.7 Summary

In this chapter, we presented the anonymous lookup protocol that provides high degree of client and server anonymity (and hence also unlinkability) in P2P networks. The protocol builds a distributed anonymity infrastructure by implementing an incentive scheme that motivate nodes to participate in the system and also maintain secrecy about the identities of nodes they interact with as part of any transaction. The protocol uses an auction protocol for trading of network resources and ensure that the rewards received by network nodes are maximized if they truthfully follow the protocol steps. Moreover, the protocol is light-weight as it does not rely on any trusted centralized entity or require specialized encryptions to be performed by the nodes.

To the best of our knowledge this is the first protocol that uses an incentive strategy to provide sender as well as responder anonymity in large-scale P2P networks. Since the underlying network model assumes that all nodes behave selfishly (and some even maliciously), we believe that the protocol is robust enough to be deployable even in a large-scale Internet setting.

CHAPTER 8. CONCLUSIONS AND OPEN PROBLEMS

In this dissertation, we proposed strategies on how resources can be shared in large Internet-scale P2P systems, assuming that individual nodes behave selfishly in a game-theoretic sense. Specifically, we proposed a model (based on the work of Nisan et. al.) for developing protocols for selfish P2P networks, and achieved the following:

Provided a solution for enabling distributed computing by harnessing idle computing resources, such as CPU cycles, in P2P networks taking into account nodes' selfishness.

Developed a mechanism such that resources like data and routing bandwidth can be priced and traded in P2P networks.

In addition, we presented a novel reputation management framework for large-scale P2P systems. Therein, we demonstrated the mechanisms for and feasibility of using reputation as a substitute for actual currency, and described how reputation computation can be carried out when nodes behave selfishly (and even maliciously). The framework can be easily extended to mimic the actual monetary transactions taking place among pair of nodes. The system of virtual currency implemented is completely distributed, robust, and light-weight. Moreover, we used game theory and its notion of Nash equilibrium to model the behavior of selfish nodes and calculate their optimum strategy for participation in a P2P network such that the problem of free-riding is minimized. The proposed game theoretic model minimizes the free-riding problem and has several significant advantages - fairness, simple implementation, and ease of calculating optimum strategies. Furthermore, as part of our effort to provide anonymity to P2P users, we developed a novel protocol to achieve both sender and receiver anonymity. The protocol is inherently anonymous, light-weight, and incentive-compatible, and is easily deployable in large-scale P2P systems.

There are several remaining open issues and problems that can be pursued in future to extend the work presented in this dissertation.

1. First, one can explore the applicability of game theory for modelling heterogeneous P2P systems and for reputation management. As P2P systems gets bigger and provide more value-added services, it seems reasonable to assume that peers would have more incentive to behave selfishly. Moreover, game theory is an ideal tool for modelling such a system with selfish users. Therefore, P2P systems appear to be a natural playing field for various game theoretic ideas and formulations. However, one must be careful when using game theory, since some of the commonly used assumptions in game theory, such as knowledge about all the players in a game, do not necessarily hold true in P2P systems.

2. Second, for systems such as CompuP2P to be commercially successful, they must provide a range of qualities of service for various parameters, such as security, fault-tolerance, cost, delay and throughput guarantees, etc. A market economy will not work if buyers do not know the quality of what they are buying, or sellers cannot ensure the quality of what they are selling. In networking, one would like various types of assurances regarding bandwidth, (end-to-end) delay, etc., and be willing to pay based on the strength of the assurance. These assurances may range from weak hints to strong guarantees.

3. Third, although we included "security" as one of the quality-of-service parameter above, in practice, security is the most important issue if P2P systems are to become pervasive and replace the existing client-server systems. Security issues for P2P systems are the same that arise in any distributed computing environment - ranging from authentication, authorization, confidentiality, privacy, anonymity, code and data misuse (as in the context of CompuP2P), malicious processing (such as virus uploads), malicious routing, etc. Although solutions (in some form at least) exist to address the above security issues, relying on technologies such as PKI, cryptography, proof carrying codes [105], digital signatures [106], etc., implementing these solutions in a completely distributed manner to be useful for P2P systems is still a real challenge.

4. Last, but not the least, it is important to understand the topologies formed by selfish nodes, specifically the factors governing such decisions, and how accurately they can be predicted. As a

simplification, the work presented in this dissertation largely assumes that nodes truthfully follow a given protocol for P2P network formation. It might be interesting to study what happens when nodes behave selfishly even while selecting their neighbors.

In conclusion, we hope that this dissertation serves as a useful step in the direction of achieving true Internet computing. The issues raised and the solutions provided should be helpful to system designers for building robust, scalable, distributed, and light-weight Internet computing architectures.

APPENDIX

We define function $d()$ to take as input two Chord IDs (or nodes) and return the Chord distance between them. In other words, $d(xy)$ returns the Chord distance between two nodes x and y.

Lemma 9. *For the lookup initiated by C, some node, say a, lies on at most a single request chain.*

Proof. Two cases can arise,

Case 1: $d(Ca) \geqslant 2^{m-1}$: Since only one lookup request is sent for all the terminal nodes that are at a Chord distance greater than 2^{m-1} from C (all these terminal nodes require going through the $m-1^{th}$ finger table entry), a can be on the lookup path to at most one terminal node.

Case 2: $d(Ca) < 2^{m-1}$: Let $2^{i-1} \leqslant d(Ca) \leqslant 2^{i-1}$, i.e., the Chord ID of a lies between the i^{th} and j^{th} finger table entry of C. a can only be on the lookup paths to terminal nodes that lie in the same range, i.e., the i^{th} and j^{th} finger table entry of C.

Without loss of generality, let a be on the lookup paths to two terminal nodes T_{R_i} and T_{R_j} $(1 \leqslant i, j \leqslant k)$, and $d(CT_{R_i}) \leqslant d(CT_{R_j})$. Then we must have that $d(CT_{R_i}) \leqslant d(Ca)$ (because the lookup path to a terminal node lying between the i^{th} finger table entry and a would not pass through a). But since at most a single request is sent out for all the terminal nodes that go through the same next hop neighbor - the terminal node selected is one which is closest to that neighbor. Therefore, C sends a request towards only T_{R_i}. Hence, we have a contradiction that a is on the lookup path to both T_{R_i} and T_{R_j}.

□

The following is a direct result of the above lemma.

Corollary 1. *For a given lookup transaction, at most a single request chain passes through any node, except for the originator of the lookup request.*

BIBLIOGRAPHY

[1] Gnutella. http://gnutella.wego.com/.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA*, July 2000.

[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *In Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *In Proceedings of ACM SIGCOMM, San Diego, CA*, 2001.

[5] A. Rowstron, and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[6] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technichal Report UCB//CSD-01-1141, Computer Science Division, University of California, Berkeley, CA*, April 2001.

[7] http://www.oreilly.com/catalog/peertopeer/chapter/ch01.html. Last accessed on June 1998.

[8] http://www-106.ibm.com/developerworks/java/library/j-p2p/. Last accessed on June 1998.

[9] Napster. http://www.napster.com/. Last accessed on July 2001.

[10] Kaaza. http://www.kaaza.com. Last accessed on January 2003.

[11] F. Kaashoek, and D. R. Karger. Koorde: A simple degree-optimal hash table. *In Proceedings of the 2nd International Peer-To-Peer Systems Workshop (IPTPS)*, 2003.

[12] A. T. Gai, and L. Viennot. Broose: A Practical Distributed Hashtable Based on the De-Bruijn Topology. *In Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing (P2P04), ETH Zurich, Switzerland*, August 2004.

[13] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *In Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[14] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[15] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic evolution of the butterfly. *In Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[16] P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A Peer-to-Peer Lookup Service Over Arbitrary Topology. *In Proceedings of IEEE INFOCOM, San Francisco, CA, USA*, 2002.

[17] C. Frey, P. J. Huffstutter, and D. Wilson. New Web Sites Clogged in Aftermath; Internet: Breakdown highlights weakness of medium not ready to compete with radio, TV. (September 12, 2001). The LA Times.

[18] SETI@home. http://setiathome.ssl.berkeley.edu. Last accessed on August 2003.

[19] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *In Proceedings of ACM SOSP 01 (Banff, Canada, 2001)*, 2001.

[20] A. Rowstron, and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *In Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[21] E. Adar, and B. A. Huberman. Free Riding on Gnutella. First Monday, vol. 5, No. 10, October 2000.

[22] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. *In Proceedings of MMCN '02*, 2002.

[23] B. Wilcox-O'Hearn. Experiences Deploying a Large-Scale Emergent Network. *In Proceedings of the First International Workshop on Peer-to-Peer Systems, ITPTS*, March 2002.

[24] N. Nisan. Algorithms for Selfish Agents: Mechanism Design for Distributed Computation. *In Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, volume 1563, Springer, Berlin*, 1999.

[25] R. Gupta, and A. K. Somani. A Framework for Reputation Management and Using Reputation as Currency in Large-Scale Peer-to-Peer Networks. *In Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing, ETH Zurich, Switzerland*, July 2004.

[26] D. A. Turner, and K. W. Ross. A Light-Weight Currency Paradigm for the P2P Resource Market. *In Proceedings of the Seventh International Conference on Electronic Commerce Research, Dallas, TX*, June 2004.

[27] P. Daras, D. Palaka, V. Giagourta, D. Bechtsis, K. Petridis, and M. G. Strintzis. A Novel Peer-to-Peer Payment Protocol. *In Proceedings of IEEE EUROCON*, 2003.

[28] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A Secure Economic Framework for Peer-to-Peer Resource Sharing. *In Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[29] M. Bellare, J. Garay, C. Jutla, and M. Yung. VarietyCash: a multi-purpose electronic payment system. *In Proceedings of the 3rd Usenix Workshop on Electronic Commerce*, August 1998.

[30] G. Medvinsky. A Framework for Electronic Currency. *PhD thesis, University of Southern California, CA, USA*, 1997.

[31] N. Asokan. Fairness in electronic commerce. *PhD thesis, University of Waterloo, Ontario, Canada*, May 1998.

[32] A. Mas-Collel, W. Whinston, and J. Green. *Microeconomic Theory.* Oxford university press, 1995.

[33] Condor. http://www.cs.wisc.edu/condor/. Last accessed on April 2005.

[34] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing, vol. 63, pp. 597-610*, 2003.

[35] M. Snir, S. Otto, S. H. Lederman, D. Walker, and J. Dongarra. MPI - The Complete Reference (vol. 1, The MPI Core), 2nd Edition, The MIT Press, 1998.

[36] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering, vol. 18, pages: 103-17*, Feb 1992.

[37] Ian Foster, and Carl Kesselman. The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition, Morgan Kaufmann, 2004.

[38] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over internet - The POPCORN project. *In Proceedings of the 18th IEEE Int. Conf. Distributed Comput. Syst., pages 592-601*, May 1998.

[39] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan. Harmony: A Desktop Grid for Delivering Enterprise Computations. *In Proc. of GRID'03.*

[40] VMware. Vmware virtual platform technology white paper. Technical report, VMware Inc., February 1999.

[41] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 654-663*, May 1997.

[42] J. Shneidman, and D. C. Parkes. Rationality and Self-Interest in Peer to Peer Networks. *In Proceedings of IPTPS 2003, Berkeley*, February 2003.

[43] D. Geels, and J. Kubiatowicz. Replica Management Should Be A Game. *In Proceedings of the SIGOPS European Workshop*, 2002.

[44] H. T. Kung, and W. Chun-Hsin. Differentiated Admission For Peer-To-Peer Systems: Incentivizing Peers To Contribute Their Resources. *In Proceedings of the 2003 Workshop on Economics of Peer-to-Peer Systems, Berkeley CA*, 2003.

[45] T. J. Ngan, D. S. Wallach, and P. Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. *In Proceedings of IPTPS 2003, Berkeley*, February 2003.

[46] S. M. Lui, K. R. Lang, and S. H. Kwok. Participation Incentive Mechanism in Peer-to-Peer Subscription Systems. *In Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), vol. 9*, January 2002.

[47] D. Hales. From Selfish Nodes to Cooperative Networks - Emergent Link-Based Incentives in Peer-to-Peer Networks. *In Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing (P2P04), ETH Zurich, Switzerland*, August 2004.

[48] T. Moreton, and A. Twigg. Enforcing collaboration in P2P routing services. *In Proceedings of the First International Conference on Trust Management*, 2003.

[49] L. Buttyan and J. P. Hubaux. Enforcing service availability in mobile ad-hoc WANs. *In Proceedings of the IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), Boston, MA*, August 2000.

[50] L. Buttyan and J. P. Hubaux. Stimulating cooperation in self-organizing mobile ad-hoc nertworks. *ACM Journal for Mobile Networks (MONET), special issue on Mobile Ad Hoc Networks*, summer 2002.

[51] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. *In Proceedings of IEEE INFOCOM, vol. 3*, March 2003.

[52] M. J. Osborne. A course in game theory. Cambridge, Massachusetts : MIT Press, c1994.

[53] M. R. Baye. Managerial Economics and Business Strategy. *Third edition, McGraw Hill*, 2000.

[54] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance, pages 8-37*, 1961.

[55] V. Sekhri. CompuP2P: A Light-Weight Architecture for Internet Computing. *Masters thesis. Iowa State University*, 2005.

[56] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. *In Proceedings of the USENIX Symposium on Interner Technologies and Systems*, December 1997.

[57] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. A Secure Environment for Untrusted Helper Applications: Confining the Wily Hacker. *In Proceedings of the 6th USENIX Security Symposium, pp. 1-13*, July 1996.

[58] Standard Performance Evaluation Corporation. *SPECjvm98 Documentation*, Release 1.0. August 1998. Online version at http://www.spec.org/osg/jvm98/jvm98/doc/index.html.

[59] ISTOS. http://ecpe.ee.iastate.edu/dcnl/DCNLWEB/Tools/tools_ISTOS.htm. Last accessed on February 2004.

[60] T. S. Eugene, and H. Zhang. Prediciting Internet Network Distances with Coordinates-Based Approaches. *In Proceedings of IEEE INFOCOM 2002, New York, NY*, June 2002.

[61] V. N. Padmanabhan, and L. Subramanian. An Investigation of Geographic Mapping Techniques for Internet Hosts. *In Proceedings of ACM SIGCOMM '01, San Diego, USA*, August 2001.

[62] I. Foster, and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *In Proceedings of International Supercomputing Applications, 11(2)*, 1997.

[63] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. *In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, 2004.

[64] A. Nikolaidis, S. Tsekeridou, A. Tefas, and V. Solachidis, V. A survey on watermarking application scenarios and related attacks. *In Proceedings of the International Conference on Image Processing*, October 2001.

[65] S. Tuomas. Limitations of the Vickrey Auction in Computational Multiagent Systems. *In Proceedings of the 2nd International conference on Multi-Agent Systems, pages 299-306. Kyoto, Japan*, December 1996.

[66] B. Felix. Cryptographic Protocols for Secure Second-Price Auctions. *In Proceedings of the Co-operative Information Agents V, Lecture Notes in Computer Science, Modena, Italy*, September 2001.

[67] A. M. Odlyzko. The history of communications and its implications for the Internet. *Available online at http://www.research.att.com/amo*, 2000.

[68] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. *In Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2003.

[69] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Managing and sharing servents' reputations in P2P systems. *IEEE Transactions on Data and Knowledge Engineering, 15(4):840–854*, July/August 2003.

[70] S. Marti, and H. Garcia-Molina. Limited reputation sharing in P2P systems. *In Proceedings of the 5th ACM conference on Electronic commerce, New York, NY, USA*, 2004.

[71] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. EigenRep: Reputation Management in P2P Networks. *In Proceedings of the ACM World Wide Web Conference, Budapest, Hungary*, May 2003.

[72] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. *In Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM '01)*, 2001.

[73] A. Farag, and M. Muthucumaru. Evolving and Managing Trust in Grid Computing Systems. *In Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering*, 2002.

[74] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative Peer Groups in NICE. *In Proceedings of IEEE INFOCOM*, 2003.

[75] S. Jordi, and S. Carles. Reputation and Social Network Analysis in Multi-Agent Systems. *In Proceedings of AAMAS '02*, July 2002.

[76] Z. Diamadi, and M. J. Fischer. A Simple Game for the Study of Trust in Distributed Systems. *Appeared in Wuhan University Journal of Natural Sciences*, 2001.

[77] Xiong Li, and Liu Ling. A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities. *In Proceedings of the 4th ACM conference on Electronic commerce (extended abstract), San Diego, CA*, 2003

[78] N. Ntarmos, and P. Triantafillou. SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks. *In Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing (P2P04), ETH Zurich, Switzerland*, August 2004.

[79] Mojo Nation. http://www.mojonation.net/. Last accessed on May 2004.

[80] B. Yang, and H. Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. *In Proceedings of CCS'03*, October 2003.

[81] http://www.socialnetworks.org/. Last accessed on February 2005.

[82] http://www.ascusc.org/jcmc/vol3/issue1/garton.html. Last accessed on February 2005.

[83] C. Buragohain, D. Agrawal, S. Suri. A Game Theoretic Framework for Incentives in P2P Systems. *In Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P'03)*, 2003.

[84] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. *In Proceedings of the 2001 ACM Conference on Electronic Commerce*, 2001.

[85] A. Pfitzman, and M. Waidner. Networks without user observability. *Journal of Computer and Security, vol. 6, no. 2*, 1987.

[86] A. Acquisti, R. Dingledine, and P. Syverson. On the Economics of Anonymity. *In Proceedings of Financial Cryptography, Lecture Notes in Computer Science, vol. 2742*, August 2003.

[87] M. K. Reiter, and A. D. Rubin. Crowds: Anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1998.

[88] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2001.

[89] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. *In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[90] L. Xiao, Z. Xu, and X. Zhang. Low-Cost and Reliable Mutual Anonymity Protocols in Peer-to-Peer Networks. *IEEE Transactions on Parallel and Distributed Systems, vol. 14, pages. 829-840*, 2003.

[91] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM, vol. 24, pages. 84-88*, 1981.

[92] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A Protocol for Scalable Anonymous Communications. *In Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

[93] Y. Guan, X. Fu, R. Bettati, and W. Zhao. An Optimal Strategy for Anonymous Communication Protocols. *In Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002.

[94] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous Connections and Onion Routing. *In Proceedings of the IEEE Symposium on Security and Privacy*, 1997.

[95] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent yet Anonymous Web Access with LPWA. *Communications of the ACM, vol. 42, no. 2, pages. 42-47*, 1999.

[96] E. Gabber, P. B. Gibbons, Y. Matias, and A. Mayer. How to Make Personalized Web Browsing Simple, Secure and Anonymous. *In Proceedings of the Conference of Financial Cryptography*, 1997.

[97] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology, vol. 1, no. 1, pages. 65-75*, 1988.

[98] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship resistant and source-anonymous web publishing system. *In Proceedings of the 9th USENIX Security Symposium*, 2000.

[99] K. Bennett, and C. Grothoff. GAP – Practical anonymous networking. *In Proceedings of the Privacy Enhancing Technologies Workshop*, 2003.

[100] S. Hazel, and B. Wiley. Achord: A variant of the chord lookup service for use in censorship resistant peer-to-peer publishing systems. *In Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[101] C. Odonell, and V. Vaikuntanathan. Information Leak in the Chord Lookup Protocol.*In Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing*, 2004.

[102] J. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. *In Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[103] K. J. Kumar, and M. Bansal. Anonymity in Chord. www.cs.berkeley.edu/ kjk/chord-anon.ps, December 2002.

[104] S. Steinbrecher, and S. Kopsell. Modelling Unlinkability. *In Proceedings of the Privacy Enhancing Technologies Workshop*, 2003.

[105] A. W. Appel. Foundational proof-carrying code. *In Proceedings of Logic in Computer Science*, 2001.

[106] C. Wong and S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Transactions on Networking, vol. 7*, 1999.

ACKNOWLEDGEMENTS

This is the part of my dissertation that I had been waiting to write for a long time. I always used to think how it feels like when one writes this section. Now that I am finally doing this, I feel very excited and relieved. As a disclosure I would like to state upfront that people, especially my family members, acknowledged here are remembered not only while I writing this section, but during every single day of my life. One might say I am being very philosophical and emotional, but I am justified in being so because of two main reasons. First, I am finishing my doctorate in "philosophy", and second, since no one is expected to reach this far in reading my dissertation (assuming if anybody does in fact undertake this daring project), I can take the privilege of saying things I normally would not say.

I would like to thanks my advisor and mentor Prof. Arun K. Somani for hid help and guidance throughout my stay at Iowa State University. His own success in his career has been a source of inspiration for me to excel in my PhD work. One of his greatest quality as an advisor is to give his students complete freedom and flexibility to explore new areas and work in fields they are most interested in. He has a very open mind and his critique and uncanny understanding of even new areas helped me to greatly improve the overall quality of my technical papers. I would also like to thank my other committee members - Prof. Tom Daniels, Prof. Akhilesh Tyagi, Prof. Srikanat Tirthapura, and Prof. Johnny Wong for their valuable suggestions and comments that greatly helped me to better define my research problem and present my solutions.

I would like to thank my friends - Souvik, Varun, Srivatsan, and Pallab, who were my constant companions during the past three years here at Ames. The time spent with them, especially in coffee room, used to be most enjoyable while on campus. I feel I have been able to make true friends whose friendship I will cherish for the rest of my life. At this point I would also like to thank my first bunch of room-mates, Srini, Abu, Murari, and Rajeev, whose support, humor, and friendship made my introduction to the PhD program a very memorable one. Srini, especially is an amazing guy whose help I will never forget.

My family is my biggest asset. There are no words to describe how lucky I am, and how thankful I am to God for giving me birth in this incredibly beautiful and great family. My grand-parents, uncles and aunts, parents, sisters, and my wife together provide me a shell in which I know I am always safe and happy. At every step of my life I had confidence in the knowledge of the fact that there are many hands to help and support me if I ever falter, or boost me if I ever am down. With time this support and love from my family has only grown stronger and fonder. I would not have been able to achieve anything in life if it had not been for the love, support, and constant motivation from my grand-parents, parents, uncles and aunts, and my sisters. My sister, Manjri, and brother-in-law, Devesh, have been my unofficial guardian for past several years now, always there for me and caring for me. My mother is a pillar of strength for me and I always look up to her for guidance and motivation. Her devotion and care for her family is simply unmatched.

My wife, Nidhi, is more than I had ever even dreamed of and stood by my side in the most difficult of times. Her love for me is endless and unconditional. Even my small stipend as a grad student seemed like a windfall, especially because of the incredible care that Nidhi took of our home and lives. Her constant smile, support, and confidence in me is what carried me through most uncertain of times as a grad student. My wife and my mother are two people who always had the greatest of belief in me, even at times when I myself did not think much about myself. They are life's anchor and my truest treasure. I work hard and pray that I can make my family proud and make their lives more comfortable always.

The research reported in this dissertation is partially supported by Jerry R. Junkins endowment at Iowa State University.