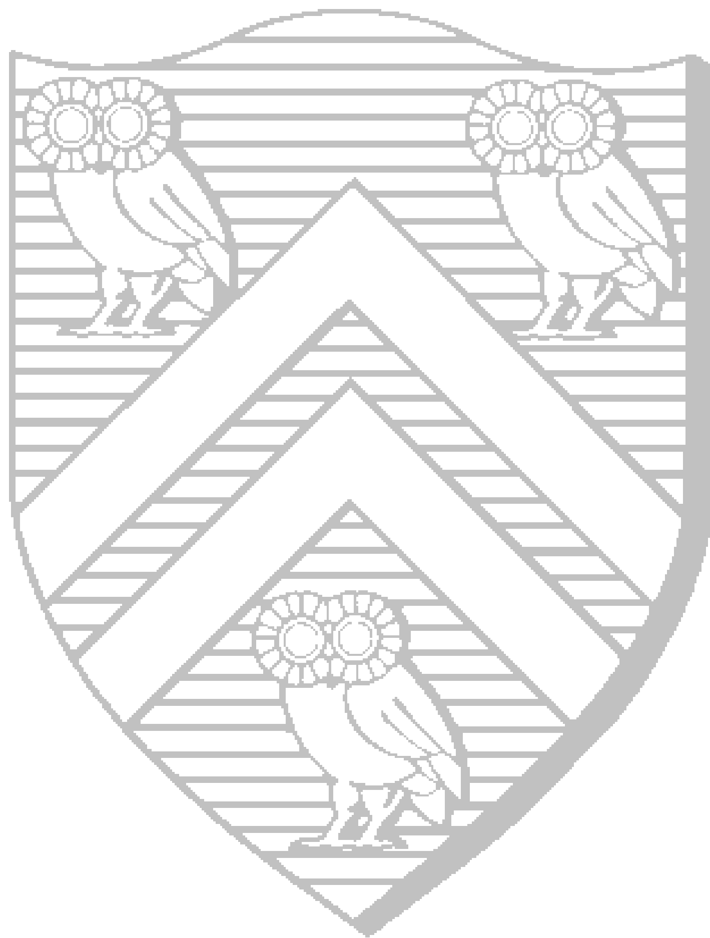
INCENTIVES AND FAIR SHARING IN PEER-TO-PEER SYSTEMS

Tsuen-Wan "Johnny" Ngan



Thesis: Master of Science
Computer Science
Rice University, Houston, Texas (March 2004)

RICE UNIVERSITY

Incentives and Fair Sharing in Peer-to-Peer Systems

by

Tsuen-Wan "Johnny" Ngan

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

Dan S. Wallach, Chair
Assistant Professor of Computer Science

Peter Druschel
Professor of Computer Science

T.S. Eugene Ng
Assistant Professor of Computer Science and
Electrical & Computer Engineering

Houston, Texas

March, 2004

Incentives and Fair Sharing in Peer-to-Peer Systems

Tsuen-Wan "Johnny" Ngan

Abstract

Cooperative peer-to-peer applications are designed to share the resources of each participating computer for the common good of everyone. However, users do not necessarily have an incentive to donate resources to the system if they can use the system's resources for free.

This thesis presents mechanisms to enforce fair sharing of limiting resources in peer-to-peer systems. Storage fairness is enforced by requiring nodes to publish their storage records and allowing auditing to those records. Bandwidth fairness is enforced by having nodes locally track the amount of data transferred and limiting each node's interactions to a small number of nodes that are proven trustworthy. Thus, a node must provide good service to receive good service. For storage systems to be efficient, nodes should provide overcapacity. Based on an economic analysis of utility functions, we show how the overcapacity parameter should be set and why clustering of the system would benefit users.

Acknowledgments

First and foremost, I think my advisor Dan S. Wallach for his guidance throughout the preparation of this thesis. His support and insightful views helped me a lot in shaping my research direction.

I am grateful to Moez A. Abdel-Gawad, Shu Du, and Khaled Elmeleegy for their work on a class project that partly stimulated the work in this thesis. I am also grateful to my colleagues Andrew Fuqua, Animesh Nandi, and Atul Singh for their input.

Last but not least, I also thank Peter Druschel and T.S. Eugene Ng for agreeing to be in my thesis committee and giving me useful reviews and suggestions, which greatly improved the quality of this thesis, despite their heavy schedules.

Contents

Abstract ii

Acknowledgments iii

List of Illustrations vii

1 Introduction 1

1.1 Motivation 1

1.2 Contributions 2

1.3 Organization 3

1.4 Background 3

1.5 Threat models 4

1.6 Secure routing and centralized authority 4

2 Storage-Constraining Systems 6

2.1 Introduction 6

2.2 Assumptions 7

2.2.1 Challenge mechanism 7

2.3 Basic designs 8

2.3.1 Smart cards 8

2.3.2 Quota managers 8

2.4 Auditing 9

2.4.1 Usage files 9

2.4.2 Attacks and audits 10

2.4.3 Extensions 12

2.4.4 Cheating patterns 13

2.5 Experiments 14
2.5.1 Results 15
2.6 Related work 18
2.7 Summary 19

3 Bandwidth-Constraining Systems 20

3.1 Overview 20
3.2 Design details 21
3.2.1 Pairwise trade 21
3.2.2 Transitive trade 22
3.2.3 Node bootstrapping 24
3.2.4 Self-adjusting debt threshold 25
3.2.5 Relationship throttling 26
3.3 Experiments 27
3.3.1 Workload model 28
3.3.2 Debt based routing with transitive trading 28
3.3.3 Static and dynamic debt thresholds 30
3.3.4 Freeloaders 30
3.3.5 Relationship throttling 33
3.4 Related work 35
3.5 Summary 37

4 Economics Behavior 38

4.1 Introduction 38
4.2 Model 40
4.2.1 Agent preferences 40
4.2.2 Successfulness function 42
4.2.3 Properties of the indirect utility function 42
4.3 Implications 43

4.3.1 Agent participation 43
4.3.2 Administration 44
4.4 Successfulness vs. overhead 45
4.5 Related work 46
4.6 Summary 46

5 Future Work **48**

Bibliography **50**

A Single-peakedness proof **57**

Illustrations

2.1 A p2p system with local/remote lists. 10
2.2 A cheating chain. 12
2.3 Overhead with different number of nodes. 16
2.4 Overhead with different number of files stored per node. 17
2.5 Overhead with different average node lifetime. 17

3.1 Using a debt-based path to leverage indirect debts to perform direct data transmission. 23
3.2 Request denial rate for different trading policies. 29
3.3 Cumulative retry distribution for requests with transitive trading using DBR and a debt threshold of 3 requests. 29
3.4 Comparing the self-adjusting debt threshold mechanism with static debt thresholds. 31
3.5 Request denial rates for good nodes vs. freeloaders. 32
3.6 Effort rates for good nodes vs. freeloaders to retrieve an object. 32
3.7 Request denial rates for good nodes and freeloaders. 34
3.8 Effort rates for good nodes and freeloaders. 34

4.1 Hard drive space usage of an agent. 39
4.2 The single-peakedness of the successfulness function. 43
4.3 Probability of successful object insertions. 45

Chapter 1

Introduction

A large number of peer-to-peer (p2p) systems have been developed recently, providing a general-purpose network substrate [MM02, RFH+01, RD01a, SMK+01, ZKJ01] suitable for sharing files [DKK+01, DR01], among other applications. In practice, particularly with widespread p2p systems such as Napster, Gnutella, or Kazaa (see, e.g., Oram [Ora01]), many users may choose to consume the resources of other users in the p2p systems without providing any of their own resources for the use of others [AH00]. Users have no natural *incentive* to provide services to their peers if it is not somehow required of them.

This thesis considers methods to design such requirements directly into the p2p system. While we could take a traditional quota enforcement approach, requiring some kind of trusted authority to give a user "permission" to store files, or even to make requests on the p2p system, such notions are hard to create in a system of decentralized peers. Why should some peers be placed in a position of authority over others? If all nodes were to publish their resource usage records, directly, where other nodes are auditing those records as a part of the normal functioning of the system, or if each node can apply fairness measures locally, we might be able to create a system where nodes have natural incentives to publish their records accurately and share their resources fairly. Ideally, we would like to design a system where nodes, acting selfishly, behave collectively to maximize the common welfare.

1.1 Motivation

In cooperative p2p applications, users are assumed to share their resources in exchange for the right to consume resources of others. The theory behind is that remote resources have higher value than local resources (fault tolerance, etc.). Participation benefits all parties if

the trade is fair. Ideally, these applications are extremely scalable, since the demand and supply of resources should both grow with the number of participants. However, in real Internet applications, it is observed that many people consume resources but do not contribute their resources for others to use. These users are referred as "free riders" or "freeloaders" and the situation "the tragedy of the commons" [Har68]. In a study of Gnutella, Adar and Huberman [AH00] found that 70% of users free ride, and half of the total requests are served by 1% of the users. Free riding is a real threat that has to be dealt with for p2p applications to maximize the benefit of all participants.

The most widely deployed p2p application to date is file sharing. Depending on the use model, the limiting resources could be different. For an archival system, storage space may be the limiting resource. For popular file sharing systems, the limiting resource is more likely the upload bandwidth. Different mechanisms are required to enforce the fair sharing of different resources. However, without a trusted central authority to enforce these mechanisms, we have to rely on the nodes themselves for the enforcement. Thus, it is also necessary to understand the economic behaviors of individual user to guarantee that rational users have the incentive to abide the rules and enforce the mechanisms.

1.2 Contributions

We investigate the fairness problem on p2p file sharing systems. The major contributions of this thesis include the following:

- Defines the threat models of resources abuse in p2p systems.
- Presents a novel decentralized auditing mechanism with low overhead to guarantee storage space fairness.
- Shows that a simple accounting scheme works well to encourage fair bandwidth sharing.
- Shows that p2p systems can be analyzed with traditional economic game-theoretic analysis techniques.

- Provides a foundation for modeling resources and preferences.
- Analyzes individual preferences and utility functions and shows how the systems can be designed and administrated to improve global utility.

1.3 Organization

This thesis contains works that appeared in two earlier publications [NWD03, FNW03], as well as some previously unpublished material. The remainder of this chapter provides necessary background of the p2p substrate *Pastry*, the p2p storage system *PAST*, and some inherit security concerns of all p2p systems. Chapter 2 discusses mechanisms to ensure fair sharing of storage in p2p systems. Chapter 3 considers fair sharing in p2p systems where bandwidth is the constraining resource. Chapter 4 analyzes an economic model of agent behavior in fair storage sharing. Related work is discussed at the end of each chapter. In Chapter 5, we conclude with some ideas for future research.

1.4 Background

The designs in this thesis are intended to work well for a wide variety of different peer-to-peer systems, although for concreteness we will describe our system in terms of Pastry [RD01a] and PAST [RD01b].

Pastry is a structured p2p overlay network designed to be self-organizing, highly scalable, and fault tolerant. In such overlays, every node and every object is assigned a unique identifier randomly chosen from a large id space, referred to as a *nodeId* and *key*, respectively. Given a message and a key, Pastry can route the message to the live node whose nodeId is numerically closest to the key in less than $\log_{16} N$ hops, where N is the number of nodes in the system.

PAST is a storage system built on top of a structured overlay and can be viewed as a distributed hash table (DHT). Each stored item in PAST is given a 160 bit key (hereafter referred to as the *handle*), and replicas of an object are stored at the k live nodes whose

nodeIds are the numerically closest to the object's handle (these nodes are called a *replica set*). PAST maintains the invariant that the object is replicated on k nodes, where k is the *replication factor*, regardless of node addition or failure. If a node in the replica set is out of space, the object will be diverted to a node close in nodeId space but not in the replica set, and stored there temporarily. The handle is built from a cryptographically secure hash (e.g., SHA-1) applied to the data being stored. As such, the handle has sufficient information for the holder of the handle to verify that the actual file has not been modified in transit.

1.5 Threat models

Before talking about our designs, it is important to first understand the threats such designs must address. We consider three adversarial models:

No collusion Nodes, acting on their own, wish to gain an unfair advantage over the system, but they have no peers with which to collude.

Minority collusion A subset of the p2p system, which may be controlled by one or multiple agents, is willing to form a conspiracy to lie about their resource usage. However, it is assumed that most nodes in the p2p system are uninterested in joining the conspiracy.

Minority bribery Like minority collusion, a subset of the nodes will form a conspiracy to lie about their usage, however, the adversary may choose specific nodes to join the conspiracy, perhaps offering them a bribe in the form of unfairly increased resource usage.

1.6 Secure routing and centralized authority

In studying routing security for p2p systems, Castro et al. [CDG$^+$02] focused only on minority collusions. Bribery would allow very small conspiracies of nodes to defeat the secure routing primitives. In this thesis, we assume the correctness of the underlying p2p

system. In particular, we assume that each user gets one node identifier that cannot be forged, and that messages sent into the p2p system will reach their correct destination.

Pastry, like other structured p2p overlays, assumes that nodeIds are assigned randomly and uniformly from the 160-bit space of possible identifiers. Attackers who can choose nodeIds can compromise the integrity of Pastry or any other structured p2p overlay. Even when they cannot choose nodeIds, they may still be able to mount "Sybil" attacks if they can obtain a large number of legitimate nodeIds easily [Dou02]. Such attacks can be prevented only by limiting the attacker's ability to join the system multiple times. Castro et al. [CDG+02] consider several approaches to accomplish this, although the only robust approach they identify requires a trusted central authority (CA) to issue entrance permits. Aside from issuing such permits, the CA is otherwise uninvolved in the operation of the p2p system, limiting the damage that can be caused if the CA is offline. The CA is assumed to serve the common good and all members of the p2p system must fully trust the CA. As such, the CA can potentially be extended for other operations requiring a globally trusted authority, but the CA's responsibilities must be limited to preserve the scalability and reliability of the p2p system.

Chapter 2

Storage-Constraining Systems

We first consider p2p storage systems, e.g., remote archival systems. In these systems, nodes store their data remotely on other nodes but rarely retrieve their data. A simple fair policy is to require each node providing the same amount of storage space it is using from the system. We consider different architectures for achieving fair storage sharing. This chapter is based on an earlier paper [NWD03].

2.1 Introduction

We describe three possible designs for storage accounting systems. These designs are targeted at p2p applications where storage space (i.e., free disk space) is the limited commodity in the system. An example of a system like this is a remote backup services, where only the writer of a given data block might want to read it (assuming the data is encrypted to preserve the writer's confidentiality). The data may go unread until the writer suffers some kind of catastrophic equipment failure.

We note that the ability to consume resources, such as remote disk storage, is a form of commodity, where remote resources have more value to a node than its local storage. When nodes exchange their local storage for others' remote storage, the trade benefits both parties, giving an incentive for them to cooperate. As such, there is no need for cash or other forms of real-world money to exchange hands; the economy can be expressed strictly in terms of single-good barter schemes in storage.

2.2 Assumptions

For all designs, we assume the existence of a public key infrastructure, allowing any node to digitally sign a file such that any other node can verify, yet it is computationally infeasible for others to forge; such an infrastructure can be provided by the CA. Because our CA's sole purpose is to assign nodeIds, it is not involved in regular p2p transactions. This reduces the cost to implement the CA, as it needs not provide highly available, scalable, or redundant service.

In our designs, it is also imperative to ensure that nodes are actually storing the files they claim to store. This is guaranteed by the following *challenge* mechanism.

2.2.1 Challenge mechanism

For each file a node is storing, it periodically picks a node that stores a replica of the same file as a target, and notifies all other replicas holders of the file that it is challenging that target. Then it randomly selects a few blocks of the file and a random key, and queries the target for a keyed hash of those blocks. The target can answer correctly only if it has the file. The target may ask another replica holder for a copy of the file, but any such request during a challenge would cause the challenger to be notified, and thus be able to restart the challenge for another file. A potential problem is that colluding nodes storing replicas of the same file could just store one copy instead of two, however this would be unlikely to occur. The nodes responsible for replicating a given file are typically constrained; in PAST, for example, they are the set of adjacent nodes with nodeIds closest to the object handle. With random nodeId assignment, few colluders would be adjacent to each other in nodeId space.

2.3 Basic designs

2.3.1 Smart cards

The original PAST paper [DR01] suggested the use of smart cards to enforce storage quotas. The smart card produces signed endorsements of a node's requests to consume remote storage, while charging that space to an internal counter. When storage is reclaimed, the remote node returns a signed message that the smart card can verify before crediting its internal counter.

Smart cards avoid the bandwidth overheads of the decentralized designs discussed later. However, smart cards must be issued by a trusted organization, and periodically reissued to invalidate compromised cards. Users must also have suitable interfaces to connect the smart cards to their computers. The higher costs of sustaining such an organization make it appears unsuitable for grass-roots p2p systems.

2.3.2 Quota managers

If each smart card was replaced by a collection of nodes in the p2p system, the same design would still be applicable. We can define the *manager set* for a node to be a set of nodes adjacent to that node in the nodeId space (i.e., a subset of that node's leaf set), making them easy for other parties to discover and verify. Each manager must remember the amount of storage consumed by the nodes it manages and must endorse all requests from the managed nodes to store new files. To be robust against minority collusion, a remote node would insist that a majority of the manager nodes agree that a given request is authorized, requiring the manager set to perform, perhaps a Byzantine agreement protocol [CL99] or possibly a k-of-n secret splitting digital signature.

The drawback of this design is that request approval has a relatively high latency and the number of malicious nodes in any manager set must be less than one third of the set size. Furthermore, managers suffer no direct penalty if they grant requests that would be correctly denied, and thus could be vulnerable to bribery attacks.

2.4 Auditing

While the smart card and quota manager designs are focused on enforcing quotas, an alternative approach is to require nodes to maintain their own records and publish them, such that other nodes can audit those records. Of course, nodes have no inherent reason to publish their records accurately. This section describes how we can create natural economic disincentives to nodes lying in their records.

2.4.1 Usage files

Every node maintains a *usage file*, digitally signed, which is available for any other node to read. The usage file has three sections:

- the *advertised capacity* this node is providing to the system;
- a *local list* of (nodeId, handle) pairs, containing the identifiers and sizes of all files that the node is storing locally on behalf of other nodes; and
- a *remote list* of handles of all the files published by this node (stored remotely), with their sizes.

Together, the local and remote lists describe all the credits and debits to a node's account. Note that the nodeIds for the peers storing the files are not stored in the remote list, since this information can be found using mechanisms in the storage system (e.g., PAST). We say a node is "under quota," and thus allowed to write new files into the system, when its advertised capacity minus the sum of the files in its remote list multiplied with the number of replications, is positive. Since the entries in local/remote lists have to be matched, all usage files have to be balanced. By increasing the advertised capacity, a node can store more files on the system, but it also has to make an equal amount of space available. By adding matched pairs in the local list of one node and the remote list of another, the credit is transferred from the latter node to the former.

When a node A wishes to store a file F_1 on another node B, first B must fetch A's usage file to verify that A is under quota. Then, two records are created: A adds F_1 to its remote

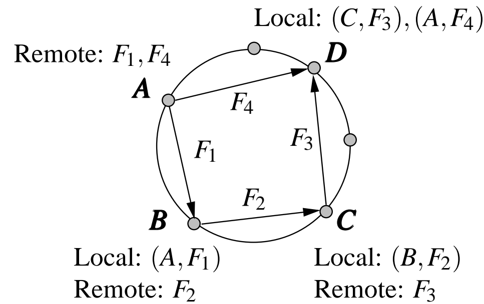


Figure 2.1 : A p2p system with local/remote lists.

list and B adds (A,F_1) to its local list. This is illustrated in Figure 2.1. Of course, A might fabricate the contents of its usage file to convince B to improperly accept its files.

2.4.2 Attacks and audits

We must provide incentives for A to tell the truth. To game the system, A might normally attempt to either *inflate* its advertised capacity or *deflate* the sum of its remote list. If A were to increase its advertised capacity beyond the amount of disk it actually has, this might allow A to consume more space than it is actually providing when the system is significantly under capacity. We believe that this is not the probable case, since providing additional space creates additional bandwidth overhead in storing files and challenges. The utility function of the users, as we will show in Section 4.2, also suggest that providing more space to the system than is required of them decreases their utility. Moreover, if the system is underutilized, nodes consuming unfair space is not a serious issue. When the utilization becomes high, A will attract storage requests that it cannot honor. A might compensate by creating fraudulent entries in its local list, to claim the storage is being used. To prevent fraudulent entries in either list, we define an auditing procedure that B, or any other node, may perform on A.

If B detects that F_1 is missing from A's remote list, then B can feel free to delete the

file.* After all, A is no longer "paying" for it. Because an audit could be gamed if A knew the identity of its auditor, anonymous communication is required, and can be accomplished using a technique similar to Crowds [RR98]. So long as every node that has a relationship with A is auditing it at randomly chosen intervals, A cannot distinguish whether it is being audited by B or any other node with files in its remote list. We refer to this process as a *normal audit*.

Normal auditing, alone, does not provide a disincentive to inflation of the local list. For every entry in A's local list, there should exist an entry for that file in another node's remote list. An auditor could fetch the usage file from A and then connect to every node mentioned in A's local list to test for matching entries. This would detect inconsistencies in A's usage file, but A could collude with other nodes to push its debts off its own books. To fully audit A, the auditor would need to audit the nodes reachable from A's local list, and recursively audit the nodes reachable from those local lists. Eventually, the audit would discover a *cheating anchor* where the books did not balance (see Fig. 2.2). Implementing such a recursive audit would be prohibitively expensive. Instead, we require all nodes in the p2p overlay to perform *random auditing*. With a lower frequency than their normal audits, each node should choose a node at random from the p2p overlay. The auditor fetches the usage file, and verifies it against the nodes mentioned in that file's local list. Assuming all nodes perform these random audits on a regular schedule, every node will be audited, on a regular basis, with high probability.

To see how frequently each node would be audited, consider a system with N nodes, where $c \ll N$ nodes are conspiring. Assume the c conspiring nodes build a cheating chain, where there is only one cheating anchor. The probability that the cheating anchor is not random audited by any non-conspiring node in one period is $\left(\frac{N-2}{N-1}\right)^{N-c}$, which approaches to $1/e \approx 0.368$ for large N. In other words, the cheating anchor would be discovered in three periods with probability over 95%.

*In practice, B should give A a grace period as A might be facing a transient failure and actually need the backup.

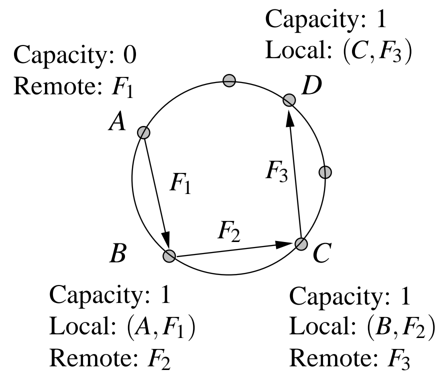


Figure 2.2 : A cheating chain, where node A is the cheating anchor with unbalanced usage file.

Recall that usage files are digitally signed by their node. Once a cheating anchor has been discovered, its usage file is effectively a *signed confession* of its misbehavior! This confession can be presented as evidence toward ejecting the cheater from the p2p system. Unlike reputation systems, the proof is non-repudiable, regardless of the credibility of the auditor. With the cheating anchor ejected, other cheaters who depended on the cheating anchor will now be exposed and subject to ejection, themselves. This would not affect non-conspiring nodes, however, as they can simply delete the involved files and make the space available for other storage requests.

We note that this design is robust even against bribery attacks, because the collusion will still be discovered and the cheaters ejected. We also note that since everybody, including auditors, benefits when cheaters are discovered and ejected from the p2p system, nodes do have an incentive to perform these random audits [FG02].

2.4.3 Extensions

Selling overcapacity. As described above, a node cannot consume more resources from the system than it provides itself. However, it is easy to imagine nodes who want to consume more resources than they provide, and, likewise, nodes who provide more resources than they wish to consume. Naturally, this overcapacity could be sold, perhaps through an

online bidding system [CGM02a], for real-world money. These trades could be directly indicated in the local and remote lists. For example, if *D* sells 1GB to *E*, *D* can write (*E*, 1GB trade) in its remote list, and *E* writes (*D*, 1GB trade) in its local list. All the auditing mechanisms continue to function.

Reducing communication. Another issue is that fetching usage logs repeatedly could result in serious communication overhead, particularly for nodes with slow net connections. To address this, we implemented three optimizations. First, rather than sending the usage logs through the overlay route used to reach it, they can be sent directly over the Internet: one hop from the target node to the anonymizing relay, and one hop to the auditing node. Second, since an entry in a remote list would be audited by all nodes replicating the logs, those replicas can alternately audit that node to share the cost of auditing. Third, we can reduce communication by only transmitting diffs of usage logs, since the logs change slowly. We must be careful that the anonymity of auditors is not compromised. For instance, a node could use version numbers to act as cookies to track auditors. To address this, the auditor needs to, with some probability, request the complete usage logs.

2.4.4 Cheating patterns

While cheating chains can be easily discovered and ejected from the system, a possible attack is for the cheating node to push back its debt to another node which, when audited, will push back its debt to another node, eventually forming a cycle instead of a chain. This might be attempted either with or without increasing the advertised capacity. We consider both cases.

Without increasing the advertised capacity, a cheating node can only balance its usage file by removing some entries from its remote list. By forcing nodes to maintain logs of changes made to their usage files (as described in Section 2.4.3), and marking each update with the (logical) timestamps of all involved parties [Lam78], we can force the cheating anchor to give a total order on all its changes. If a node claims to remove an entry from its

remote list after it has exceeded its quota, its log would be a signed confession that it has cheated. Otherwise, it has to withdraw a file from the system before storing new files. This eliminates any benefit the node might get, for itself, by participating in a cheating chain.

By increasing its advertised capacity without bound, a node might be able to gain additional storage credit in the system, hoping to take advantage of the system operating far below its actual capacity. While a freeloading agent might temporarily benefit from abusing other nodes' excess capacity, these freeloaders will be discovered and ejected as the system's free space decreases and they eventually cannot support the local storage demand of them. Thus, when legitimate nodes need space, freeloaders will be naturally pushed out of the way.

If the system is operating at or near its capacity, then any report of increased capacity on one node will immediately attract more files to be stored on that node. A node might try to simultaneously increase its published capacity and its list of locally stored files, claiming to be full; the nodes responsible for replicating these objects will notice the discrepancy.

2.5 Experiments

In this section, we present some simulation results of the communication costs of the quota managers and the auditing system. For our simulations, we assume all nodes are following the rules and no nodes are cheating. Storage spaces are chosen from truncated normal distribution, while file sizes are chosen from Pareto distribution[†] with shape parameter $\alpha = 1.3$. The storage space of each node is chosen from 2 to 200GB, with an average of 50GB. We varied the average file size across experiments. In each day of simulated time, 1% of the files are reclaimed and republished. Two challenges are made to random replicas per file a node is storing per day.

[†]Empirical measurement showed that WWW file size distribution is heavy-tailed, similar in spirit to Pareto distribution [CB96]. The bandwidth consumed for auditing is dependent on the number, rather than the size, of files being stored. We also performed simulations using real file system traces and obtained similar results.

For quota managers, we implemented Castro et al.'s BFT algorithm [CL99]. BFT is a state machine replication algorithm that tolerates Byzantine faults provided fewer than $1/3$ of the replicas are faulty, and provides safety and liveness in asynchronous environments like the Internet. We configured each manager set to have ten nodes, allowing BFT to tolerate up to three faulty nodes in the manager set. With larger manager sets, we could tolerate more faults, at a cost of more communication. For auditing, normal audits are performed on average four times daily on each entry in a node's remote list and random audits are done once per day. We simulated both with and without caching usage files, and included all the other optimizations described in Section 2.4.3.

Our simulations include per-node overhead for Pastry-style routing lookups as well as choosing one node, at random, to create one level of indirection on audit requests. The latter provides weak anonymity sufficient for our purposes. Note that we only measure the communication overhead due to storage accounting. In particular, we exclude the cost of p2p overlay maintenance and storing/fetching of files, since it is not relevant to our comparison. Unless otherwise specified, all simulations are done with 10,000 nodes, 180 files stored per nodes, and an average node lifetime of 14 days.

2.5.1 Results

Figure 2.3 shows the average upstream bandwidth required per node, as a function of the number of nodes (the average required downstream bandwidth is identical). The per-node bandwidth requirement is almost constant, thus all systems scale well with the size of the overlay network.

Figure 2.4 shows the bandwidth requirement as a function of the number of files stored per node. The overheads grow linearly with the number of files, but for auditing without caching, it grows nearly twice as fast as the other two designs. If a p2p storage system is used for large files, such as might be expected for backup systems, this overhead would be inconsequential. Nodes can be incentivized to store a smaller number of larger files by defining a minimum file size. In the event that bandwidth does eventually become a con-

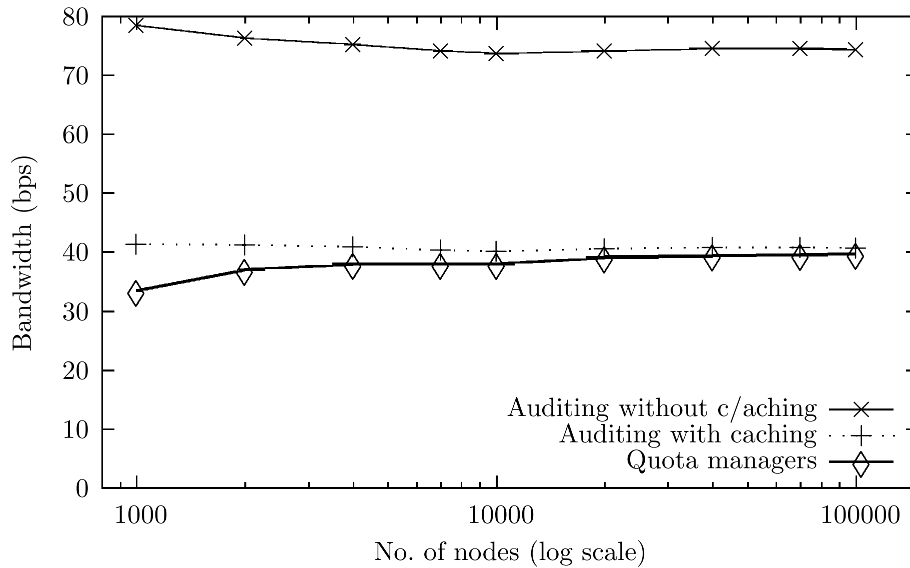


Figure 2.3 : Overhead with different number of nodes.

cern, storage might no longer be a bottleneck. Such situations are discussed in Chapter 3.

Figure 2.5 shows the overhead versus average node lifetime. The overhead for quota managers grows rapidly when the node lifetime gets shorter, mostly from the cost in joining and leaving manager sets and from approving file insertions for nodes newly joined the system. While most Internet-based file-sharing systems have relatively short node lifetimes, we would expect that a p2p storage system would use maintained office machines rather than home computers, and thus have longer node lifetimes. Long lifetimes will be necessary, in any case, to avoid having normal use of the system dominated by the bandwidth costs of maintenance and replication [BR03].

Our simulations have also shown that quota managers are more affected by the file turnover rate, due to the higher cost for voting. Also, the size of manager sets determines the vulnerability of the quota manager design. To tolerate more malicious nodes, we need to increase the size of manager sets, which would result in a higher cost.

In summary, auditing with caching has a very low overhead, linear in the number of files stored and scaling well as the number of nodes in the system grows. Relative to the

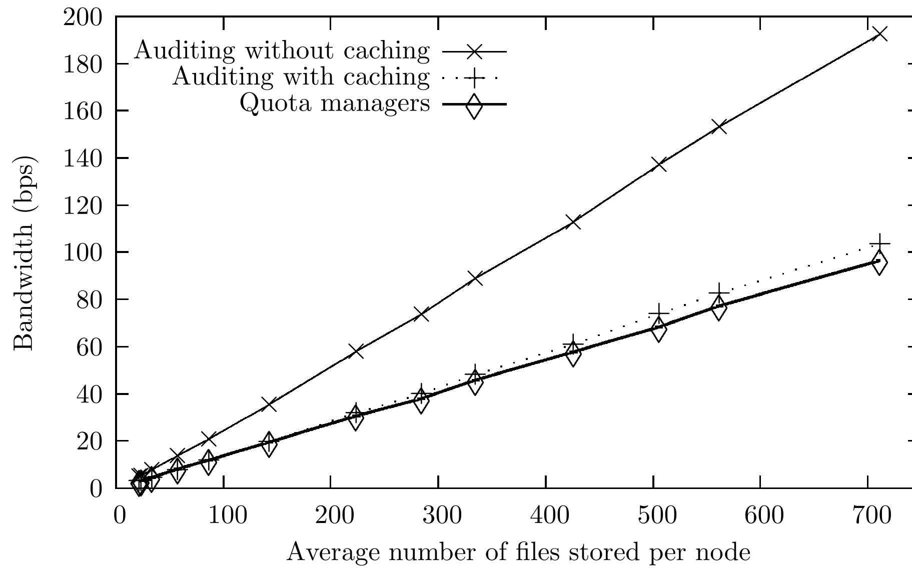


Figure 2.4 : Overhead with different number of files stored per node.

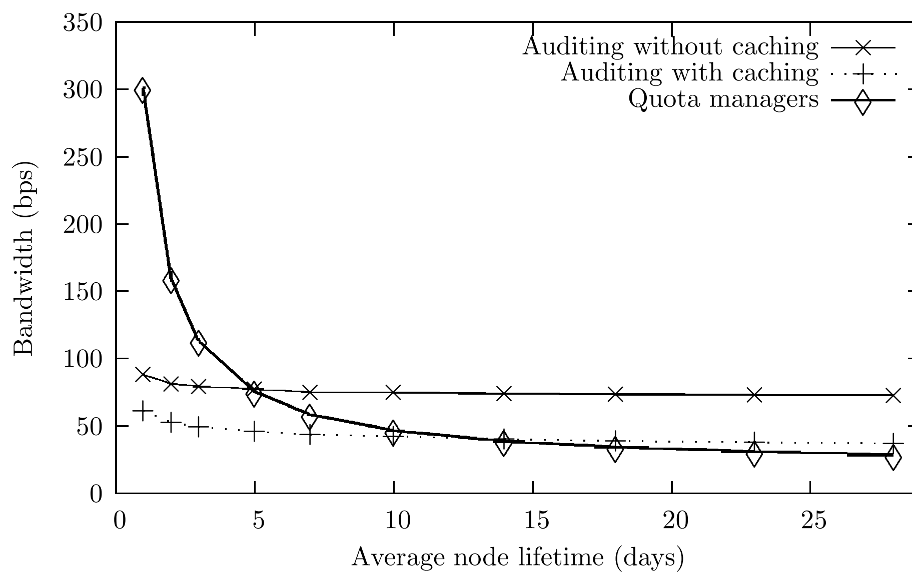


Figure 2.5 : Overhead with different average node lifetime.

bandwidth required for storing and retrieving files, the auditing overhead is of the order of tens of bps, only a small fraction of a typical p2p node's bandwidth that would participate on an archival system. Auditing provides a practical and bandwidth-efficient mechanism to ensure fair sharing in storage-constrained systems, providing resistance to malicious nodes and scalability to large p2p systems.

2.6 Related work

Storage sharing

Tangler [WM01] is designed to provide censorship-resistant publication over a small number of servers (i.e., less than 30), exchanging data frequently with one another. To maintain fairness, Tangler requires servers to obtain "certificates" from other servers which can be redeemed to publish files for a limited time. A new server can only obtain these certificates by providing storage for the use of other servers and is not allowed to publish anything for its first month online. As such, new servers must have demonstrated good service to the p2p system before being allowed to consume any system services.

Palimpsest [RH03], Tangler, and earlier systems like Gnutella and FreeNet all provide *ephemeral* storage. The p2p system makes no guarantees that a file will be available indefinitely. While popular files can be widely replicated, unpopular files will disappear for lack of interest and would then need to be reinserted. This contrasts with our own work, where inserted files can live forever.

Samsara [CN03] enforces fairness by charging peers storage space in the form of a claim, which can be replaced by real data when needed. The transfer of claims doubles the space and bandwidth required to store data. Claims can be forwarded to form chains or cycles to improve space efficiency. Since cycles can only be formed by chance, chains can grow to $O(N)$, where N is the size of the overlay network (they witnessed a chain consisting of over $3/4$ of the nodes in their experiment). In this case, a single failure could cause a majority of nodes to lose data, casting doubts on the scalability of the system.

Cooper and Garcia-Molina considered p2p trading for a small number of storage sites and proposed algorithms on peer selection so as to increase global reliability [CGM02b]. Subsequently they described how auction and bidding could work, and examined policies for deciding when to call an auction and how much to bid [CGM02a].

Economics

Fehr and Gachter's study considered an economic game where selfishness was feasible but could easily be detected [FG02]. When their human test subjects were given the opportunity to spend their money to punish selfish peers, they did so, resulting in a system with less selfish behaviors. This result helps justify that users will be willing to pay the costs of random audits. See also Chapter 4 where we model the economics behavior of agents participating in p2p storage systems.

2.7 Summary

This chapter has considered three architectures for achieving fair storage sharing of resources in p2p systems. Experimental results indicate that auditing has small overheads and is scalable to large numbers of files and nodes. In practice, auditing provides incentives, allowing us to benefit from its increased resistance to collusion and bribery attacks.

Chapter 3

Bandwidth-Constraining Systems

In this chapter, we consider the problem of p2p systems where many users wish to download objects, and bandwidth, particularly for the nodes providing the objects, is the limiting resource. If users can consume the system's resources without providing any of their own, they absolutely will. For example, Adar and Huberman found that 70% of Gnutella nodes were sharing no objects, and nearly 50% of responses were served by 1% of the hosts [AH00]. While Chapter 2 discussed how we can provide incentives to *store* data, we also need to provide incentives to *serve* it, thereby avoiding freeloading situations.

This chapter contains joint work with Animesh Nandi and Atul Singh.

3.1 Overview

We assume there are two types of nodes. *Good nodes* fetch objects and follow the system's design to serve them. *Freeloaders* also fetch objects, but never serve any. Nodes do not know in advance the type of each other. The goal of our design is to worsen the service received by freeloaders without effecting that of good nodes.

Fundamentally, there are two simple constants that a node can measure between itself and every node with which it has a *relationship*: the number of objects (or bytes or some other globally-specified block size) sent and the number of objects received. The difference of these two numbers says something about the *debt* or *credit* that a node has with its peer. Likewise, the total number of objects that a node has received from a peer measures the *confidence* that a node has in its peer. As such, two nodes that have been sharing objects for a long time might have low debt and high confidence in each other. Note that confidence is not symmetric. In particular, when a freeloader exploits a good node, the confidence of

the good node in the freeloader would be low, whereas the reverse would be high.

A low-overhead accounting mechanism would have every node maintain a table of all nodes with which it has exchanged data. For each entry, a node maintains the debt/credit and the confidence values it has in its peers. No distributed auditing of these pairwise debt lists is necessary, as every node is maintaining this information locally for its own benefit. Likewise, there exists no opportunity for a freeloader to introduce fraudulent data into other nodes' tables. If a node is asked to transmit an object to a receiver who is already deeply in debt, the node may refuse to service that request.

3.2 Design details

We choose to update the debt and confidence values only when a node fetches an object from another node. In other words, we count data traffic but not control traffic (e.g., request forwarding). While refusing data traffic can be a sensible thing to do when the sender has no incentive, refusing control traffic could partition or otherwise reduce the reliability of the p2p overlay. If a node refuses to forward any control traffic, its peer may conclude that the node is unresponsive, and simply remove it from the p2p overlay. A full study of incentives in p2p control traffic is beyond the scope of this thesis, although Daswani et al. [DGM02] examined mechanisms to limit the effectiveness of using flooding queries for denial-of-service attacks.

3.2.1 Pairwise trade

The debt and confidence values can be used by good nodes to discriminate freeloaders from other good nodes, allowing them to refuse service to freeloaders. An obvious policy would be setting a *debt threshold*. Requests would be honored unless the debt would go beyond the debt threshold. This debt threshold might be increased dynamically as a function of the confidence value, giving more slack to peers that have performed well over time.

In general, a node has no incentive to serve an object to a node with which it has no prior relationship. Imagine a mature node (having lived in the system for a long time) *A*

that has served many objects to other nodes, and thus many nodes will be indebted to A and would honor requests from A. However, if A wants to read an object from a node Z, who does not happen to owe A anything, how can A leverage the credit it already has to obtain the object? We solve this problem by *transitive trading*.

3.2.2 Transitive trade

Transitive trade allows a node to take advantage of its earned credit to obtain objects from nodes which might otherwise refuse to serve it. It works by identifying a *debt-based path* from itself to a node that has the desired object. In a debt-based path, each node in the path has credit with the next node it has a relationship with, and is below the debt threshold of the next node. Locating such debt-based path can be done by *debt-based routing*.

Debt-based routing (DBR). P2p substrates like Pastry support *locality-based routing* based on an arbitrary locality metric, typically related to the network delay measured between any two nodes, to achieve low latency paths. DBR uses debt instead of delay to choose the next routing hop. Such a route will not necessarily be optimal in terms of minimizing the network latency from source to destination, but it will increase the likelihood of finding a debt-based path to realize a transitive trade. The debt-based routing table, which is used to support DBR, consists of all the nodes with which the local node has a relationship.

If the debt-based routing table does not contain an entry for a given prefix, our DBR lookup will instead choose the most appropriate node from the remaining nodeIds in the table. This has the potential to result in routes longer than $O(\log N)$. Our simulations indicate that longer routes are rarely used. Furthermore, missing entries in the routing table will cause some DBR lookups to fail to identify a path from source to destination. The system will retry the DBR lookups by taking a different first routing hop to a node with which the local node has a relationship. Our simulations indicate a small number of retries are sufficient to satisfy most requests. Section 3.3.2 discusses these simulation results in more detail.

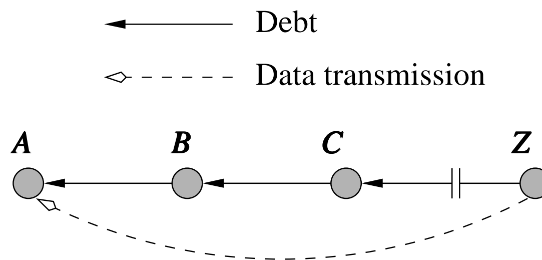


Figure 3.1 : Using a debt-based path to leverage indirect debts to perform direct data transmission.

In theory, once we have identified one of these debt-based paths, we could conceivably rearrange all the debts such that the destination node now owes something not to its predecessor in the route, but instead to the source of the route. This is illustrated in Figure 3.1. A series of debts, where B owes A, C owes B, and so forth until Z owes its predecessor could be replaced with a direct debt from Z to A. Every node in the chain has an incentive to perform this debt swap, since a debt to a neighbor could be forgiven without actually transmitting any bulk object data over the network.

Transitive trade protocol. To make debt swapping work, we need a protocol that is robust against any node in the trading chain cheating, attempting to cancel a debt that it owes without giving up the debt owed to it by the next node in the trading chain. Rather than trying to perform some kind of complex cryptographic commitment protocol, we take an incremental approach. We assume that Z has an object that A wishes to download. This object is large enough to be decomposed into a large number of smaller fixed-size blocks, perhaps on the order of 50–100 kbytes. A routes a message through the trading chain to Z requesting a specific block. Z transmits the block directly over the underlying network to A. A then transmits subsequent block requests along the same trading chain. Intermediate nodes can incrementally adjust their debt and credit tables when they forward these requests.

If any party in the trade refuses to pass along the control traffic, then the data traffic will stop as well; the party dropping the request will get at most one "free" block. A

might choose to keep several requests outstanding, perhaps to fetch non-overlapping blocks concurrently from multiple replicas, with the risk of incurring more loss if some nodes in the p2p overlay drop their control traffic. If A pays a bounty to the intermediate nodes, perhaps spaced out over the duration of its download from Z, the incentive to collect the bounties will outweigh the incentive to cheat the protocol.

Debt-based path length. It is preferable to have short debt-based paths, as they are more efficient in terms of both time and bandwidth. Moreover, such paths would be less vulnerable to malicious nodes disrupting the transitive trade. Therefore, it will be worthwhile to make numerous probes through the debt-based routing tables to identify such paths. Greater searching effort will result in more efficient trades at the expense of additional control traffic.

3.2.3 Node bootstrapping

When a new node joins the system, it has no debts and no credits. Unless other nodes in the system offer some form of altruism, nobody will be willing to honor requests from this new node to read an object. To address this, we see two possible approaches: limited altruism or probationary period with limited privileges. Our design borrows concepts from both approaches.

Systems like BitTorrent [Coh03] altruistically allocate a limited amount of every node's bandwidth to answer requests, regardless of the debt a node may have accumulated. While this altruism helps bootstrap a system, it also creates opportunities for freeloading that we wish to avoid. By having a dynamically adjusted threshold (see Section 3.2.4), we hope to have the benefits of BitTorrent's altruism without its opportunity for freeloading.

Systems like Tangler [WM01] force new nodes into a probationary mode where they service the requests of other nodes, but have no opportunity to make their own requests for system services. Tangler requires all data in the system to be republished every two weeks and was never meant to scale to support millions of nodes. However, we can achieve

a similar probationary effect for very little cost by leveraging the replication that already occurs in p2p storage systems like PAST. When a new node joins the system, it is *required* to replicate a set of objects where it is a member of that object's replica set. The pre-existing members of the replica set have an incentive to replicate their home objects to the new node, providing the load-balancing and availability guarantees normally associated with p2p systems. Object replication becomes the new node's avenue to acquire credit toward several nodes which might be requesting those objects. It can then leverage these credits to get objects that it needs. The recent study on Kazaa traces showed that the popularity of objects is often short lived and the most popular objects tend to be recently born ones [GDS+03]. Thus, object replications would give a chance to new nodes to store newly born and probably more popular objects, thereby accumulate some credits for it to start requesting objects.

3.2.4 Self-adjusting debt threshold

Limited altruism allows new nodes to bootstrap while ensuring that freeloaders only get a limited number of objects. However, it raises the concern of whether the performance of good nodes will also be effected. Debt thresholds may result in a good node's request being denied by another good node due to a temporary asymmetry in the request distribution. Setting a high debt threshold would alleviate this problem, yet it could potentially be exploited by freeloaders.

More formally, we must set the debt threshold to minimize the probability that a good node is denied of service by another good node. The distribution of requests between two good nodes, if random, should resemble the distribution of heads versus tails in random coin flips [Han91].

Imagine a fair coin is tossed $2n$ times. Let D be the absolute difference between the number of heads and tails obtained. The probability distribution of D is given by

$$P(D=2k) = \begin{cases} \left(\frac{1}{2}\right)^{2n} C_n^{2n} & k=0 \\ 2\left(\frac{1}{2}\right)^{2n} C_{n+k}^{2n} & k=1,2,\ldots \end{cases}$$

The expectation value is

$$\langle D_n \rangle = \frac{n \times C_n^{2n}}{2^{2n-1}} \ ,$$

which can be closely approximated by $2\sqrt{n/\pi}$. Essentially, this means that the expected "debt" between any two nodes (the difference between the number of requests in two directions) should be proportional to the square root of the total requests between them. For large n, the debt can be closely approximated by a normal distribution with standard deviation proportional to $\sqrt{n}$.

Each coin toss can be considered to be an object request between a pair of good nodes, with the coin determining which side made the request. Over time, the expected value of the debt would still grow proportionally to the square root of the number of requests. This implies that no matter how high we set the debt threshold, it is expected that any constant debt threshold between good nodes would be exceeded over time.

Instead, we set the debt threshold proportional to the square root of the confidence value, compensating for these expected debts. Good nodes can still obtain objects from each other even if they have incurred small debts, building up a mutual trust over time, yet freeloaders will not benefit much from the limited altruism. By adjusting the constant factor, we can limit the probability of any one pair of nodes exceeding the debt threshold to arbitrarily small.

3.2.5 Relationship throttling

Unfortunately, as the number of nodes in the p2p system grows, the amount of available altruism might still be sufficient for freeloaders to exploit. Limited altruism, alone, is insufficient to prevent freeloaders. We also need some mechanism to limit the rate at which a freeloader can exploit the available altruism in the system by continuously forming new relationships.

Mature nodes with excess credit can easily find debt-based paths to any other node. As such, they have no incentive to accumulate more credit by altruistically sending objects to nodes with which the sender has no debt. The incentives are much different for new

nodes. Unlike mature nodes, new nodes *want* to acquire credit to redeem later; they have an incentive to allow other nodes to fetch objects from them.

More generally, we can define the *quality of service* (QoS) that a node experiences to be equal to the inverse of the running average of the number of retries necessary for its object fetch requests to succeed (more retries implies lower QoS). When a node's QoS is high, it will have no interest in accepting a request from a node with which it does not have pre-existing relationship. When a node's QoS is low, it wants to accept new connections.

Using such a QoS metric, good nodes will start out accepting all connections. Once their QoS becomes satisfactory, they can subsequently refuse new connections. Freeloaders will only be able to get service from good nodes during the brief window while they are new but have not yet reached a desirable QoS level.

3.3 Experiments

In this section, we present some simulation results of our bandwidth-constraining design. Nodes may go offline, but they maintain their debt-based routing table when they return again. Objects are replicated using PAST's replication strategy. When an object request is rejected, nodes will retry for a fixed number of times, each time using a different route. Each node also has a fixed, 1024-object soft cache to store objects it has previously requested. This cache can be used by the node to serve any request it sees for those objects. Each node also maintains the debt-based routing table, containing nodes with which the node has dealt with in the past. The debt-based routing tables are persistent across node departures, i.e., a node remembers its table when it returns into the system, and other nodes remember a node even if it has left the system for a while.

In this section, we try to answer the following questions:

- How effective is debt-based routing with transitive trading?
- How does the self-adjusting debt threshold compare with static debt threshold?
- How effective are debt thresholds against freeloaders?

- How effective is relationship throttling?

3.3.1 Workload model

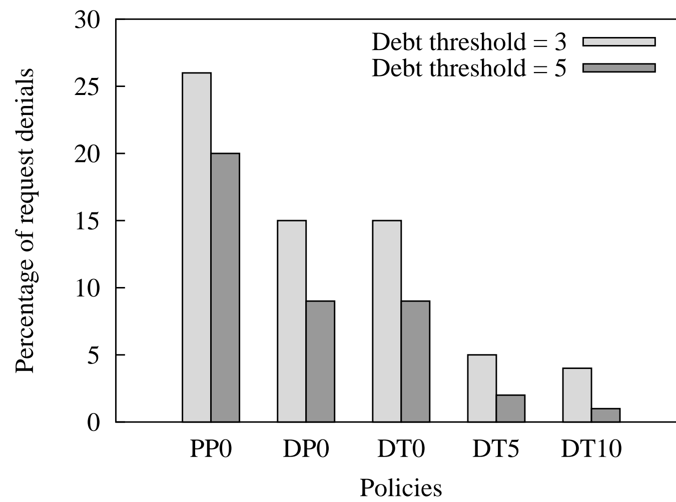
P2p file sharing systems represent the class of systems in which bandwidth is the constraining resource. We use the model described by Gummadi et al. [GDS+03] to generate workloads. This model, derived from Kazaa traces, captures the fetch-at-most-once behavior and the importance of new objects and new clients in typical p2p file sharing applications.

Based on this model, we choose the following parameters: number of nodes online $C = 800$, number of objects $O = 40,000$, request rate per node $\lambda_R = 50$, object arrival rate $\lambda_O = 12$, and node arrival rate $\lambda_C = 5$ (the units are nodes or objects per unit of simulated time). Each object is initially replicated to three nodes. We assume that there is a fixed pool of 1,000 distinct nodes, out of which 800 are online at any time. Nodes that go online/offline are chosen randomly. Each simulation considers about 200 units of simulated time.

3.3.2 Debt based routing with transitive trading

We first evaluate the effectiveness of debt-based routing, transitive trading, and retries. For a system consisting of only good nodes, we fix the debt threshold to 3 and 5 requests, and measure the fraction of requests denied. Figure 3.2 shows the results of five different policies. As expected, debt-based routing, increasing debt thresholds, and allowing failed requests to be retried all increase the odds of a request ultimately succeeding. Combining all three techniques gives the greatest benefit.

Debt-based routing appears to offer a clear benefit, but increased debt thresholds and allowing request retries have costs associated with them. Larger debt thresholds give more opportunity for freeloaders to take advantage of the system. We discuss techniques to limit this in Section 3.3.3. Request retries generate additional messaging traffic in the p2p system. Figure 3.3 shows how increasing numbers of retries have diminishing returns on the odds of successfully retrieving an object. Although 5% of the requests are not served



Policy	Routing	Trading	Retries
PP0	Proximity-based	Pairwise	0
DP0	Debt-based	Pairwise	0
DT0	Debt-based	Transitive	0
DT5	Debt-based	Transitive	5
DT10	Debt-based	Transitive	10

Figure 3.2 : Request denial rate for different trading policies.

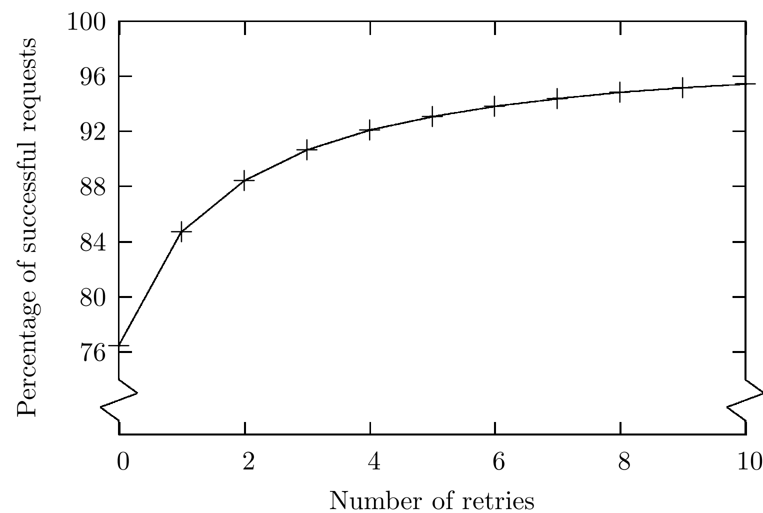


Figure 3.3 : Cumulative retry distribution for requests with transitive trading using DBR and a debt threshold of 3 requests.

even after 10 retries, nearly 90% are satisfied within three retries.

From the results, it appears that transitive trading does not reduce request denial rate at all. However, by increasing the number of attempts and relaxing the hop selection in DBR, we observed that DBR is able to reduce the denial rate from 15% to 4% (for 10 retries and DT = 3). From Figure 3.3, we also observe that DBR converges fast in finding debt-based paths as after 4 attempts, more than 90% of requests are successful.

Although relaxing the hop selection in DBR can cause increased path lengths, we observe that the average path length was only 2.1. The path length distribution was as follows: 21.6% of paths were of length 1, 54.5% of length 2 and 22.2% of length 3. The maximum observed debt-based path length was 11. Note that if we had used normal Pastry routing instead of DBR, for Pastry with $N = 800$ nodes, the expected path length would be approximately 2.26.

3.3.3 Static and dynamic debt thresholds

Section 3.2.4 showed, analytically, that debt thresholds should grow with the square root of the confidence that nodes have in their peers rather than being fixed constants. To verify this, we experimentally compared constant debt thresholds of 1, 3, and 5 requests with a dynamic mechanism. Figure 3.4 shows the request denial rates for these schemes. A low debt threshold clearly causes a high request denial rate. The dynamic threshold scheme causes an initially higher failure rate than the larger fixed thresholds, as would be expected, yet as the simulation progresses, the failure rate of the dynamic scheme seems to grow slower than the static schemes. Note that it is not a reasonable choice to set the debt threshold to an arbitrarily high constant, as it could be exploited by freeloaders.

3.3.4 Freeloaders

Next, we will introduce freeloaders into our simulation. Freeloaders make requests based on our model, similar to good nodes, but they always refuse to serve objects. We initially ran our simulations for 40 time units with only good nodes. Subsequently, we introduced

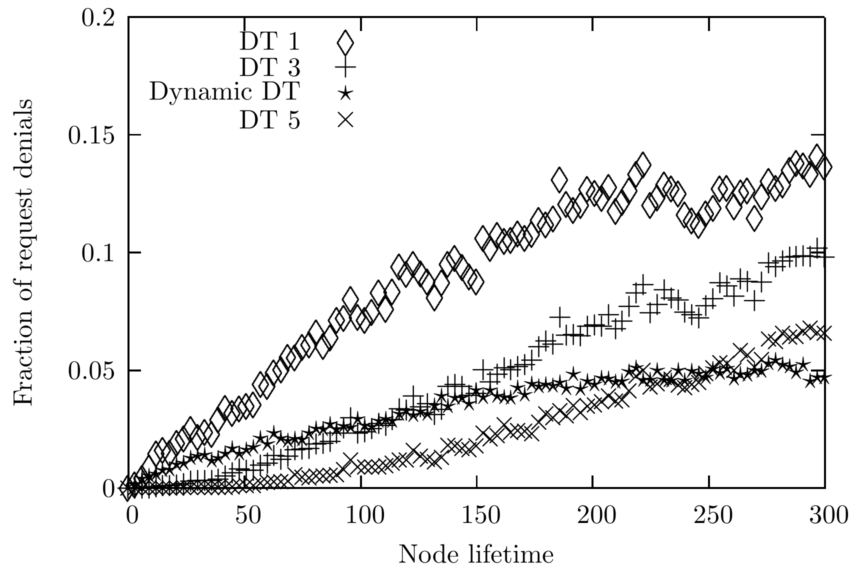


Figure 3.4 : Comparing the self-adjusting debt threshold mechanism with static debt thresholds. DT x represents setting a fixed threshold of x.

freeloaders to the system. We assume that freeloaders follow the protocol to forward control traffic. Unlike good nodes, however, freeloaders are always online throughout the entire simulation period to make requests. Recall that DBR tables are persistent across node departures, hence a freeloader cannot escape bad service by periodically departing from the system or exploit the limited altruism of good node whenever they rejoin the system after being offline for a while.

Figure 3.5 compares the request denial rate for good nodes to freeloaders over time. We observe that the failure rate starts very low for all nodes. However, after about 20 units of simulated time, the freeloaders begin to experience increasing failure rates. After only 60 units of simulated time, freeloaders will find the majority of their requests rejected by the p2p system. At the same time, good nodes experience a consistently low failure rate.

In addition to denial rates, we also measured the *effort spent*, which is defined as the average number of requests sent to successfully retrieve an object. Figure 3.6 compares the effort spent of good nodes and freeloaders. We observe that good nodes experience a stable effort spent of approximately 2 (i.e., on average, good nodes succeed after one

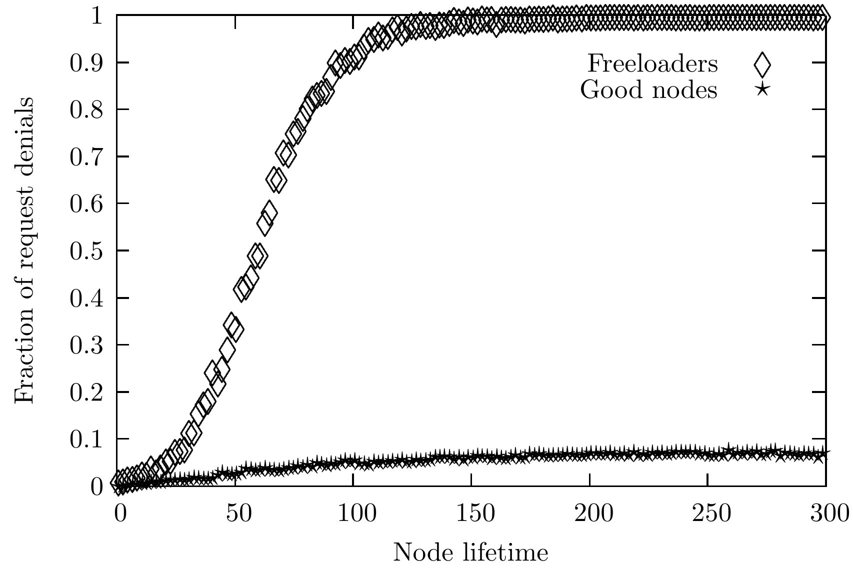


Figure 3.5 : Request denial rates for good nodes vs. freeloaders.

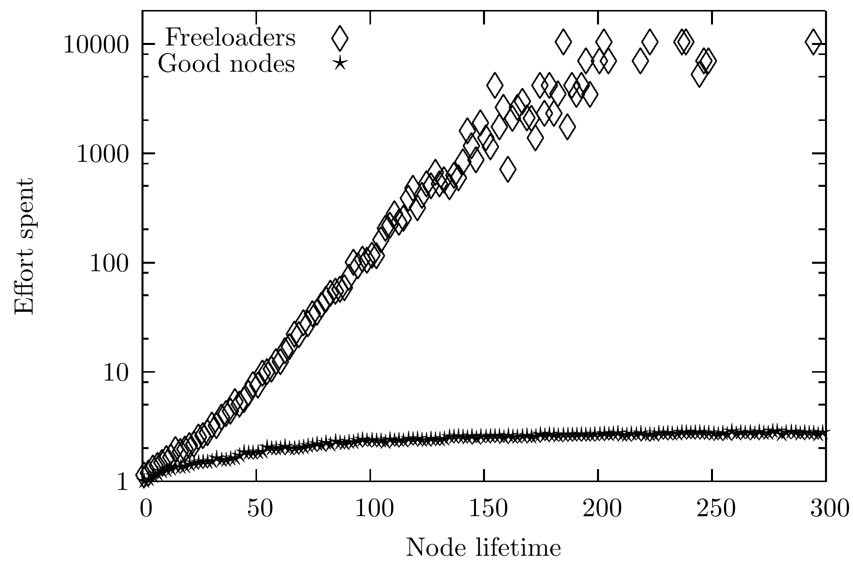


Figure 3.6 : Effort rates for good nodes vs. freeloaders to retrieve an object.

retry). The effort spent for freeloaders rises exponentially in the simulation time. Note that as the request denial rate approaches 1, the effort spent metric approaches infinity, since the number of objects retrieved approaches 0. These points are omitted in Figure 3.6.

Our results show that freeloading requires a rapidly increasing effort to achieve diminishing returns, while still maintaining a high quality of service for good nodes. While we assume uniform request rate in the simulation, it is obvious that if freeloaders increase their request rate, their request denial rates and their effort spent would increase even faster. Debt-based routing and dynamic debt thresholds can be quite effective in limiting the benefits of freeloading. Freeloaders still get some benefit when they join the system. However, freeloaders still get some benefit when they join the system. We could tune the system to guarantee that all nodes receive degraded service quality when they join the p2p system, with the quality improving only after the new node has proven its worth. To accomplish this, we will use relationship throttling to further limit the effectiveness of freeloading.

3.3.5 Relationship throttling

Relationship throttling, described in Section 3.2.5, limits the number of relationship a node maintains. To evaluate its effectiveness, we again run our simulations, adding freeloading nodes after 40 time units. By this time, the existing good nodes will have developed mature relationships with one another and will generally refuse service to new nodes.

Figure 3.7 compares the request denial rate for good nodes and freeloaders on systems with and without relationship throttling. When the freeloaders join the system without relationship throttling, their requests experience similar failure rates to good nodes for the first 20 units they are present. However, with relationship throttling, the freeloaders immediately experience a higher request denial rate that is significantly worse than the non-throttled case. (Around time 60 or so, they cross each other, but the denial rates are sufficiently high that their differences are insignificant.) Conversely, relationship throttling has very little effect on good nodes. The request denial rate curves for good nodes, with and without relationship throttling, are indistinguishable.

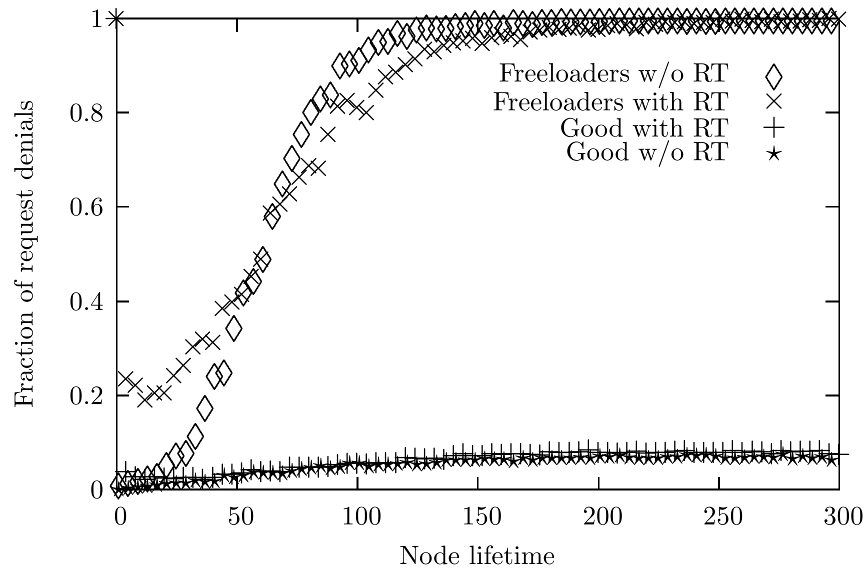


Figure 3.7 : Request denial rates for good nodes and freeloaders, with and without relationship throttling.

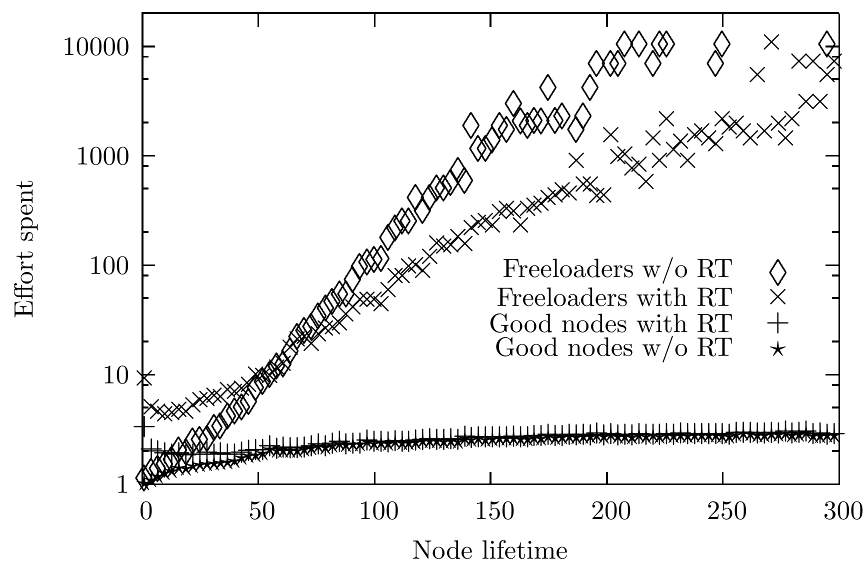


Figure 3.8 : Effort rates for good nodes and freeloaders, with and without relationship throttling.

In another experiment, we added good nodes, rather than freeloading nodes, on time 40. Within a few time units, their request denial rates dropped to within half a percent of the mature good nodes. This experiment showed that, merely by performing the replication required by PAST, new good nodes can rapidly be accepted into the p2p system.

Finally, we measured the effort spent for good nodes and freeloaders, both with and without relationship throttling. Figure 3.8 shows freeloaders experience increased effort spent from the moment they join the system, with exponential growth in effort spent over the time of the simulation. Thus, relationship throttling requires freeloaders to spend more effort, and is effective even under a Sybil attack. The effort required of good nodes, on the other hand, is barely affected. Relationship throttling does increase the initial effort required by good nodes, but it is not significantly higher than the steady state effort.

3.4 Related work

Bandwidth sharing

BitTorrent [Coh03] facilitates large numbers of nodes all trying to acquire exactly the same file. Every BitTorrent node will have acquired some subset of the file and will trade blocks with other nodes until it has the whole file and can leave the system. In order to bootstrap new nodes into the system, nodes reserve $1/4$ of their bandwidth for altruistic service. Nodes that fairly trade their bandwidth will experience a higher quality of service.

GNUnet [Gro03] uses the idea of locally-maintained debit/credit relations in similar fashion to our own work. It also has a similar idea for taking advantage of debt relationships across nodes, comparable to our debt-based routing. As they are more concerned with anonymity than network efficiency, they do not support transmitting objects directly across the network. All traffic goes through the overlay, forcing intermediate nodes to carry the bulk traffic of the object transfer while giving them no particular incentive to do this, save for maintaining their own anonymity.

NICE [LSB03] uses a reputation system where nodes build trust with one another, and

use flooding to search for trust chains to learn the reputation of other nodes. Apart from the efficiency issues with flooding, one trusted node can declare arbitrarily many new nodes to be trusted, potentially resulting in the collapse of the system.

Reputation and payments

Resource allocation and accountability problems are fundamental to p2p systems. Dingledine et al. [DFM01] surveyed schemes for reputation systems and micro-payments.

In reputation systems, if obtaining a new identity is free or cheap, while positive reputation would still be valuable, negative reputation could be shed easily. Friedman and Resnick [FR01] study the case of cheap pseudonym, and argue that suspicion of strangers is costly.

Distributed reputation systems have been proposed under the context of, among others, MIX-Net [DFHM01], Gnutella [CDdV$^+$02], and mobile ad-hoc networks [BL02]. Our approach is different from these systems because trust is based purely on locally observable (and thus more trustworthy) information.

Golle et al. [GLBML01] considered centralized p2p systems with micro-payment. They proposed several payment mechanisms and analyzed how various user strategies reach equilibrium within a game theoretic model.

The Eternity Service [And96] includes an explicit notion of electronic cash, with which users can purchase storage space. Once published, a file cannot be deleted, even if requested by the publisher. Salem et al. [SBHJ03] proposed a micro-payment architecture for multi-hop cellular networks.

KARMA [VCS03] keeps track of the resource usage of each participate in the system by a *bank-set* formed by other nodes, similar to our quota managers approach. However, since nodeIds are obtained by generating private keys, an attacker can control its bank-set simply by generating sufficient nodeIds offline.

3.5 Summary

We have presented three mechanisms: dynamic debt thresholds, transitive trade with debt-based routing, and relationship throttling. These mechanisms together make bandwidth-limited p2p file sharing systems robust against freeloaders. While good nodes will experience only modest increases in effort relative to p2p systems without fairness enforcement mechanisms, freeloading nodes will experience exponentially increasing effort and rapidly falling success rates, despite their additional work. Taken together, these mechanisms provide strong incentives for nodes to follow the rules rather than trying to freeload on the system.

Chapter 4

Economics Behavior

P2p systems introduce a new area of interplay between computer science and economics. Designers of such systems must firmly understand the incentives, preferences, and decision space of participating agents in order to decide the policies and make the system function as well as possible. This chapter considers the preferences of users in a p2p storage system on how storage might be allocated and reserved among nodes. This model would provide insight on how users would participate in the system and, more importantly, how the system should be designed to suit the users' preferences.

This chapter is based on joint work with Andrew C. Fuqua [FNW03].

4.1 Introduction

P2p networks provide a new platform for distributed applications, allowing users to share their computational, storage, and networking resources with their peers to the benefit of every participant. Most p2p system designs focus on traditional systems problems including scalability, load-balancing, fault-tolerance, and so forth. However, many systems tend to assume that all users in the system are running "official" software, whereas users have a clear self-interest in modifying their software if it allows them to consume the network's resources without contributing any of their own. P2p systems must be designed to take user incentives and rationalities into consideration [FS02, Pap01, SP03]. Given rational user behavior, we can then study the overall economic behavior of such systems. How will these systems evolve over time? How should the parameters of the system, such as the degree of object replication, be chosen when different users have different ideas about these parameters' optimal values?

H_i: | L_i | S_i | $N_i = k_i S_i$ | $R_i = \lambda k_i S_i$ |

H_i:	total hard drive space of the agent
L_i:	agent's private local space
S_i:	local copies of files the agent stores in the network
N_i:	reciprocal space the agent uses for locally storing remote files
k_i:	agent's replication factor
R_i:	additional local space the agent is required to contribute
λ:	overhead rate

Figure 4.1 : Hard drive space usage of an agent.

While such parameters could be declared by a central authority, the total utility of the system could possibly be increased if individual utility is taken into account. We address this problem by considering a game theoretic model of p2p storage networks. We consider the preferences, utility functions, and constraints of the agents in the model. This allows us to analyze the economic behavior of the agents and suggest policies for system administrators. Such a model is most relevant to p2p systems, such as distributed backup systems (e.g., Pastiche [CN02]), where storage, not network bandwidth, is the limiting resource.

Similar to traditional file systems, p2p storage performance degrades when the system is operating near its full capacity. Specifically, the probability of successfully inserting a file on the system decreases. However, unlike traditional file systems, users cannot simply purchase larger disks for their local computer to increase system capacity; they must somehow convince remote computers to allocate more space for remote storage. We accomplish this by requiring all nodes to reserve a portion of their storage space to lower the global utilization rate. Chapter 2 considers architectures to meet that goal. Building on those results, we consider the properties of the economic system that develop when "cheating" has been rendered technically infeasible.

4.2 Model

We begin modeling an agent of a node i by partitioning its hard drive space as shown in Figure 4.1 to implement a "fairness" policy. Each agent may choose its own *replication factor* k_i, while the *overhead rate* λ is a system-wide constant. Our agent wishes to store S_i units of data in the network. In reciprocity, the agent must make available $N_i = k_i S_i$ units of space for the use of remote nodes plus an additional overhead (or overcapacity) $R_i = \lambda k_i S_i$. Thus, including L_i units of private, unshared data, the total disk space usage is

$$\begin{aligned} H_i &= L_i + S_i + N_i + R_i \\ &= L_i + (1 + k_i(\lambda + 1))\, S_i \end{aligned} \tag{4.1}$$

The constant λ defines an important aspect of how the system will behave. Higher values of λ increase the efficiency of finding a node with free space to absorb storage requests, at the cost of lower effective capacity in the p2p storage network.

4.2.1 Agent preferences

Initially, we assume that all agents consider λ to be an exogenous variable. The agent then needs only to decide how much of its personal data should be archived on the network (and, thus, how much space it must make available for remote storage). Thus, an agent primarily cares about L_i, S_i (see Figure 4.1) and $p_i(\lambda)$: the probability of successfully storing a file. While an agent's preferences might vary with changes in other values, λ is the only value that must be agreed by all agents. Therefore, we will focus on the importance of λ. We start by modeling the utility function of node i to be $U_i(L_i, S_i, p_i(\lambda))$. We expect U_i to be a three good Cobb-Douglas utility function because the utility-maximizing values of the arguments should all be non-zero.* For instance, an extremely large amount of completely unreliable remote space (i.e., $p_i(\lambda) = 0$) would be useless for backups. In general, zeros

*For background reading in economics, consult the utility theory or consumer behavior section of a microeconomics textbook such as Varian [Var92, Ch. 7].

for any argument to the utility function represent degenerate cases (e.g., when an agent is providing no space for remote storage) that are uninteresting in practice.

From equation (4.1), $L_i = H_i - c_i S_i$ where $c_i = 1 + k_i(\lambda + 1)$. We can now write the utility function as

$$\begin{aligned} U_i\left(L_i, S_i, p_i(\lambda)\right) &= L_i^{\alpha} S_i^{\beta} \left(p_i(\lambda)\right)^{\gamma} \qquad \text{where} \quad \alpha + \beta + \gamma = 1 \\ &= \left(p_i(\lambda)\right)^{\gamma} \left(\left(H_i - c_i S_i\right)^{\alpha} S_i^{\beta}\right) \qquad (4.2) \\ \log\left(\left(H_i - c_i S_i\right)^{\alpha} S_i^{\beta}\right) &= \alpha \log\left(H_i - c_i S_i\right) + \beta \log S_i \qquad (4.3) \end{aligned}$$

Since $\left(p_i(\lambda)\right)^{\gamma}$ is a constant with respect to the other arguments, maximizing equation (4.2) or (4.3) will also maximize U_i. The first order conditions for maximizing equation (4.3) imply that

$$\frac{\beta}{S_i} = \frac{\alpha c_i}{H_i - c_i S_i} \quad .$$

By rearranging the terms, we have

$$\frac{\beta}{S_i} = \frac{(\alpha + \beta) c_i}{H_i} \quad ,$$

and thus

$$S_i = \frac{\beta H_i}{c_i(\alpha + \beta)} \qquad \text{and} \qquad L_i = \frac{\alpha H_i}{\alpha + \beta} \quad . \qquad (4.4)$$

We can now express an agent's preferences through an indirect utility function $v_i(\lambda)$, which gives the highest utility available to an agent for a given λ. By substituting (4.4) into (4.2), we obtain

$$v_i(\lambda) = \left(p_i(\lambda)\right)^{\gamma} \left(\frac{\alpha H_i}{\alpha + \beta}\right)^{\alpha} \left(\frac{\beta H_i}{\alpha + \beta}\right)^{\beta} \left(\frac{1}{1 + k_i(\lambda + 1)}\right)^{\beta} .$$

All terms without λ are constant and can thus be collectively represented by m.

$$v_i(\lambda) = \frac{\left(p_i(\lambda)\right)^{\gamma}}{\left(1 + k_i(\lambda + 1)\right)^{\beta}} \cdot m \qquad (4.5)$$

Before further analyzing the properties of v_i we will model the successfulness function $p_i(\lambda)$.

4.2.2 Successfulness function

Recall that $R_i = \lambda k_i S_i$ and the space R_i provides a buffer which helps maintain free space that is approximately uniformly distributed throughout the network. It is clear that for a constant S_i, a larger buffer (or equivalently a larger λ) lowers the failure rate when storing objects in a p2p storage network, assuming that file insertion is considered as failed after a fixed number of retries. Section 4.4 shows experimentally that the distributions of the successfulness function can be approximated by

$$p_i(\lambda) = 1 - e^{-t\lambda}$$

for some fixed parameter t, where t depends negatively on the average size of the files stored. As the amount of free space in the system decreases, larger files will become more difficult to store than smaller files. Conversely, as λ increases, the probability of success will increase, but with diminishing returns. Other variables will affect the general successfulness, including the replication factor k_i and how many times a failed storage request might be retried before the system aborts the request. As a general rule, successfulness can always be increased at a cost of additional storage and/or communication overhead.

4.2.3 Properties of the indirect utility function

Now that we have a model for $p_i(\lambda)$, we can examine the properties of $v_i(\lambda)$ as in equation (4.5). We show that preferences with respect to λ are single-peaked. In other words, $v_i(\lambda) = (1 - e^{-t\lambda})^{\gamma}/(1 + k_i(\lambda + 1))^{\beta}$ has a single maximum.

In Figure 4.2 we plot $v_i(\lambda)$ against λ with different parameters. It provides only visual evidence of the single-peakedness of $v_i(\lambda)$. A formal proof is shown in Appendix A.

Figure 4.2 was plotted using varied parameter values, but for most reasonable values, the peaks of v_i occur for $\lambda \in [0.3, 1.5]$. When γ was given an extraordinarily high value, the peak of $v_i(\lambda)$ shifted well to the right. Since γ is the weight in the utility function for the probability of successfully storing a file, it follows that the utility-maximizing λ for nodes desiring to store remote files easily would be larger than the λ for nodes that

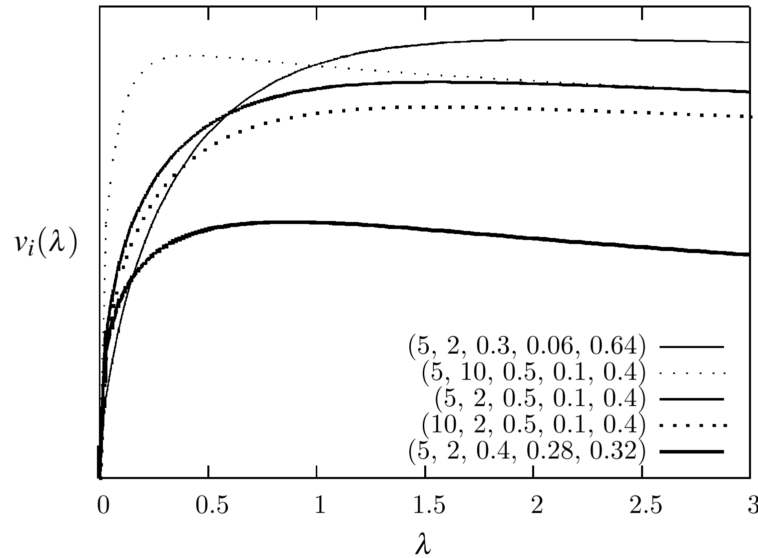


Figure 4.2 : The single-peakedness of $v_i(\lambda)$ with different values of $(k_i, t, \alpha, \beta, \gamma)$.

primarily value having more space available for local storage. We also observe that the indirect utility function is always strictly concave up to the peak, which means that there is decreasing marginal utility of λ.

4.3 Implications

In this section, we use our model to reason about rational behaviors for both agents and administrators of p2p storage systems.

4.3.1 Agent participation

Assuming that several storage networks exist, each with possibly differing λ_s values, agents will join the storage network whose parameters best suit the agent's own preferences. It is unlikely that an agent will find a λ_s which exactly equals its own desirable λ^*, but with several storage systems to choose from, agents can evaluate and rationally choose to participate in the system closest to their preferences. (The alternative, of course, is to refuse to

participate.)

As such, agents with similar preferences for λ will tend to band together. By clustering in this fashion, agents contribute their resources to form a system with the desired level of successfulness and, in effect, create a public good. For example, an agent with a preference for low overhead (and successfulness) would not join a high overload network because the agent would rather allocate its disk space for its private use rather than to the extra reserve space mandated by the higher λ. In other words, disutility is created by joining a non-optimal network. If the disutility is large enough the agent will refuse to participate at all, thus creating a market for another storage system that better suits the needs of that agent.

4.3.2 Administration

P2p systems are fundamentally designed to limit or entirely remove the role of centralized administration. Regardless, the presence of such administration can help provide a rendezvous point for new agents to determine which of many existing storage networks best suit the agent's preferences.

We also argue that a central administrator can conduct surveys of agent preferences, allowing for one or more networks to be defined to best match the expressed preferences of individual agents. If only a single system is to be established, economic voting principles suggest that the median of the agents' revealed preference variables should be chosen by the administrator, as this implies that no majority wants to increase or decrease the chosen value [Var92, Section 23.6]. Agents will truthfully reveal their preferences if they know the administrator employs this policy [Var95]. If the administrator finds that establishing clusters would benefit participants, it can partition the voting agents roughly according to their preferences and choose the median of the revealed preferences of the agents in each partition. Truthful revelation increases the likelihood that the administrator will define a storage system closer to that agent's optimal preferences, whereas lying means the agent might find itself assigned to a storage system that actually increases its disutility.

Of course, once agents join a system and begin exchanging data with one another, a

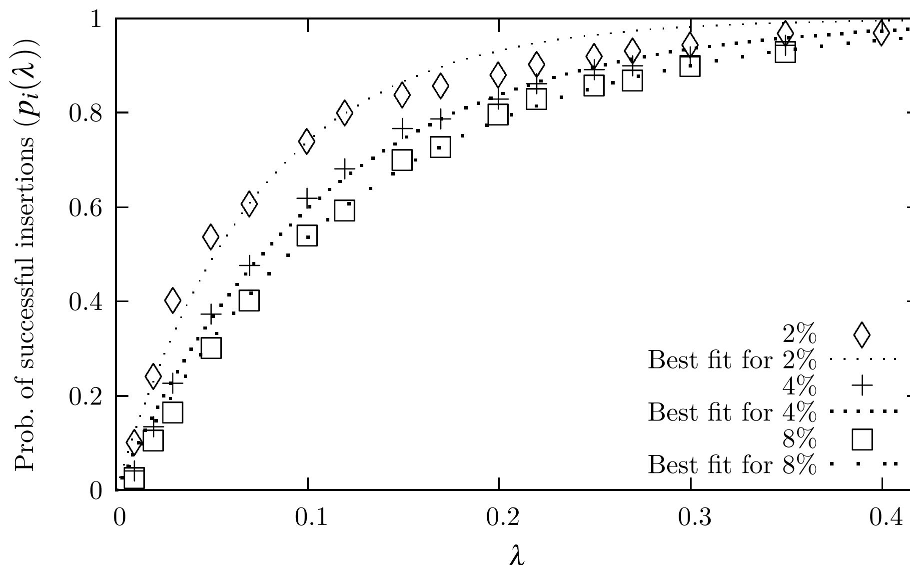


Figure 4.3 : Probability of successful object insertions with different file sizes and varying λ values. The curves are of the form $1-e^{-t\lambda}$, fitting the data points by varying t. The percentage indicates the size of the file to be inserted as a fraction of the average storage capacity one agent contributes.

mechanism such as the auditing described in Chapter 2 becomes necessary to guarantee that no agent is freeloading. The central administrator does not need to be involved in this auditing process, except perhaps acting as a "court" to which a node might bring evidence of another node's freeloading behavior.

4.4 Successfulness vs. overhead

In section 4.2, we modeled storage successfulness $p_i(\lambda)$, as a function of the overhead rate λ. This section presents simulation results to determine the exact shape of $p_i(\lambda)$.

We constructed a PAST system with 10,000 nodes, each contributing storage space chosen from a truncated normal distribution from 2 to 200GB, with an average of 50GB. First, each node stores as many file as their quota permits. Then we attempt to store additional files into the overlay network and record the probability of an eventual storage success. The result, as well as the best fit curves of the form $1-e^{-t\lambda}$, are shown in Figure 4.3. As

expected, the figure shows that $p_i(\lambda)$ increases with smaller file sizes and higher λ. It also shows that $1 - e^{-t\lambda}$ is a close approximation of $p_i(\lambda)$ measured by our simulations.

4.5 Related work

The field of Mechanism Design (MD) has existed formally for thirty years. The goal is to design the rules of interaction between agents so that their selfish, or self-interested, behavior produces some outcome deemed desirable by the designer. Classically, MD has been applied to auction theory, among other economic systems. More recently, as the view of computers as agents has become more prevalent, MD has also been applied in computational settings [Var95].

Nisan and Ronen [NR99] applied MD to solve some problems that might arise from agents manipulating algorithms to serve their own interest. Distributed Algorithmic Mechanism Design [FS02] applies MD specifically in a distributed setting and has as goals both computational tractability and incentive compatibility. It has been used to solve network problems related to multicast transmissions [FPS01], efficient routing [FPSS02], and most recently p2p systems [SP03].

We described a voting process where agents reach agreement on parameters for their shared system. The game-theoretic aspect of voting is an active research area for both economics and artificial intelligence. Voting and decision-making of distributed agents is discussed in Sandholm [San99].

Golle et al. [GLBML01] modeled centralized p2p systems with small incremental payments between agents. They proposed several payment mechanisms and analyzed how various user strategies reach equilibrium within a game theoretic model.

4.6 Summary

This chapter presents an economic model of the resources and preferences of agents in p2p storage networks. By analyzing the model, specifically the indirect utility function, we

observe that an agent has a single-peaked preference for the storage overhead rate λ. This implies that agents with similarly optimal λ values will have an incentive to cluster together and to reveal their preferences to a centralized administrator who can orchestrate this clustering. We expect this clustering will also work for other system parameters, including the degree of object replication.

Chapter 5

Future Work

This thesis has described a number of fair sharing mechanisms in p2p file sharing systems. Similar mechanisms are also needed for other applications. One application we plan to look at is p2p multicast. While it is in some sense similar to the bandwidth-constraining problem in file sharing systems, counting debts probably would not work well on a static multicast overlay topology. A more application-specific mechanism may be required.

A more fundamental incentives problem is the p2p substrates themselves. P2p systems are made possible by relying on peers forwarding messages for each other. However, due to malice or self-interest, it is entirely possible that some peers may choose to not follow the messaging passing protocol completely. In particular, they may refuse to forward messages that they cannot benefit from. Secure routing [CDG+02] circumvents this problem by detecting failures and employing redundancy. It would be desirable, however, if the p2p substrates can be designed in such a way that forwarding messages is made incentives-compatible.

Conclusions. We have presented architectures for ensuring that nodes in p2p systems have economic incentives to share their storage and bandwidth resources with their peers in order to satisfy their own needs. Our storage incentives solution, requiring nodes to perform random audits on each other's published resource usage lists, has extremely low bandwidth overhead and scales nicely to large systems. Our bandwidth incentives solution only requires nodes to track their individual pair-wise debts and uses routing, based on dynamically varying debts, with a retry mechanism, to guarantee a high likelihood of success in finding a node willing to transmit a desired file, while simultaneously limiting

the ability of freeloaders to benefit from the altruism of good nodes. In combination, these mechanisms can allow p2p systems of arbitrary scale to ensure that all nodes in the system contribute resources for the benefit of all. By analyzing the economics model of the resources and preferences of individuals, we show how one can solicit and choosing global parameters to adjust the system and improve the overall utility.

Bibliography

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[And96] Ross Anderson. The Eternity service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology*, pages 242–252, Prague, Czech Republic, October 1996.

[BL02] Sonja Buchegger and Jean-Yves Le Boudec. Performance analysis of the CONFIDANT protocol. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Lausanne, Switzerland, June 2002.

[BR03] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, Kauaii, Hawaii, May 2003.

[CB96] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 160–169, Philadelphia, PA, May 1996.

[CDdV+02] Fabrizio Cornelli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboshi, and Pierangela Samarati. Choosing reputable servents in a p2p network. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Security for structured peer-to-peer overlay networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[CGM02a] Brian F. Cooper and Hector Garcia-Molina. Bidding for storage space in a peer-to-peer data preservation system. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[CGM02b] Brian F. Cooper and Hector Garcia-Molina. Peer to peer data trading to preserve information. *ACM Transactions on Information Systems*, 20(2), April 2002.

[CL99] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[CN02] Landon P. Cox and Brian D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[CN03] Landon P. Cox and Brian D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[DFHM01] Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. A reputation system to increase MIX-Net reliability. In *Proceedings of the 4th International Workshop on Information Hiding*, Pittsburgh, PA, April 2001.

[DFM01] Roger Dingledine, Michael J. Freedman, and David Molnar. Accountability. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter 16. O'Reilly & Associates, 2001.

[DGM02] Neil Daswani and Hector Garcia-Molina. Query-flood DoS attacks in Gnutella. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Canada, October 2001.

[Dou02] John R. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[DR01] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, Schoss Elmau, Germany, May 2001.

[FG02] Ernst Fehr and Simon Gachter. Altruistic punishment in humans. *Nature*, 415(6868):137–140, January 2002.

[FNW03] Andrew C. Fuqua, Tsuen-Wan "Johnny" Ngan, and Dan S. Wallach. Economic behavior of peer-to-peer storage networks. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[FPS01] Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1), August 2001.

[FPSS02] Joan Feigenbaum, Chistos Papadimitriou, Rahul Sami, and Scott Shenker. A BGP-based mechanism for lowest-cost routing. In *Proceedings of the 21st*

ACM Symposium on Principles of Distributed Computing, New York, NY, 2002.

[FR01] Eric Friedman and Paul Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.

[FS02] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, Atlanta, GA, September 2002.

[GDS+03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[GLBML01] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge. Incentives for sharing in peer-to-peer networks. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, Tampa, FL, October 2001.

[Gro03] Christian Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Wirtschaftsinformatik*, June 2003.

[Han91] M. B. Handelsman. Distributing “heads” minus “tails”. *The College Mathematics Journal*, 22:444–446, 1991.

[Har68] Garrett Hardin. The tragedy of the commons. *Science*, 162, 1968. Alternate location: `http://dieoff.com/page95.htm`.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), July 1978.

[LSB03] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee. Cooperative peer groups in NICE. In *Proceedings of the 22nd IEEE Infocom*, San Francisco, CA, March 2003.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[NR99] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, Atlanta, GA, May 1999.

[NWD03] Tsuen-Wan "Johnny" Ngan, Dan S. Wallach, and Peter Druschel. Enforcing fair sharing of peer-to-peer resources. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, February 2003.

[Ora01] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Sebastopol, CA, March 2001.

[Pap01] Christos Papadimitriou. Algorithms, games, and the Internet. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 1–5, Hersonissos, Crete, Greece, July 2001.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, November 2001.

[RD01b] Antony Rowstron and Peter 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*, pages 188–201, Chateau Lake Louise, Banff, Canada, October 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM*, pages 161–172, San Diego, CA, August 2001.

[RH03] Timothy Roscoe and Steven Hand. Palimpsest: Soft-capacity storage for planetary-scale services. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, Kauaii, Hawaii, May 2003.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[San99] Tuomas Sandholm. Distributed rational decision making. In Gerhard Weiß, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 5. The MIT Press, Cambridge, MA, 1999.

[SBHJ03] Naouel Ben Salem, Levente Buttyán, Jean-Pierre Hubaux, and Markus Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Annapolis, MD, June 2003.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM*, San Diego, CA, August 2001.

[SP03] Jeff Shneidman and David Parkes. Rationality and self-interest in peer to peer networks. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, February 2003.

[Var92] Hal R. Varian. *Microeconomic Analysis*. W.W. Norton & Company, New York, NY, 3rd edition, March 1992.

[Var95] Hal R. Varian. Mechanism design for computerized agents. In *Proceedings of the 1st Usenix Workshop on Electronic Commerce*, New York, NY, July 1995.

[VCS03] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. KARMA: A secure economic framework for p2p resource sharing. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[WM01] Marc Waldman and David Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, November 2001.

[ZKJ01] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area address and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

Appendix A

Single-peakedness proof

In this appendix we prove the following theorem on the single-peakedness of $v_i(\lambda)$.

Theorem. *For positive constants k_i, t, β, and γ, the function*

$$v_i(\lambda) = \frac{(1-e^{-t\lambda})^\gamma}{(1+k_i(\lambda+1))^\beta}$$

is single-peaked w.r.t. λ for $\lambda \geq 0$.

Proof. We prove by showing that for $\lambda \geq 0$, the sign of the first derivative of $v_i(\lambda)$ changes exactly once, and it is from positive to negative. For notational convenience, let $z = k_i + 1$. Then

$$\begin{aligned}
v_i(\lambda) &= \frac{(1-e^{-t\lambda})^\gamma}{(k_i\lambda+z)^\beta} \\
\frac{dv_i(\lambda)}{d\lambda} &= \frac{(k_i\lambda+z)^\beta\gamma(1-e^{-t\lambda})^{\gamma-1}te^{-t\lambda} - (1-e^{-t\lambda})^\gamma\beta(k_i\lambda+z)^{\beta-1}k_i}{(k_i\lambda+z)^{2\beta}} \\
&= \frac{(1-e^{-t\lambda})^\gamma}{(k_i\lambda+z)^\beta}\left[\frac{\gamma te^{-t\lambda}}{(1-e^{-t\lambda})} - \frac{\beta k_i}{(k_i\lambda+z)}\right] \\
&= v_i(\lambda)\left[\frac{\gamma te^{-t\lambda}}{(1-e^{-t\lambda})} - \frac{\beta k_i}{(k_i\lambda+z)}\right] \\
&= v_i(\lambda)\left[\frac{\gamma t}{(e^{t\lambda}-1)} - \frac{\beta k_i}{(k_i\lambda+z)}\right] \\
&= \frac{v_i(\lambda)}{(e^{t\lambda}-1)(k_i\lambda+z)}\left[\gamma t(k_i\lambda+z) - \beta k_i(e^{t\lambda}-1)\right]
\end{aligned}$$

Let $\Phi(\lambda) = \gamma t(k_i\lambda+z) - \beta k_i(e^{t\lambda}-1)$. Since $v_i(\lambda)$, $(e^{t\lambda}-1)$, and $(k_i\lambda+z)$ are all positive, we only need to show that $\Phi(\lambda)$ changes its sign exactly once, and the sign changes from positive to negative. First we note that $\Phi(0) = \gamma tz > 0$. The derivative of $\Phi(\lambda)$ is

$$\Phi'(\lambda) = k_i t(\gamma - \beta e^{t\lambda}) \ .$$

Solving $\Phi'(\lambda) = 0$ for λ, we obtain $\lambda = \frac{1}{t}\ln(\frac{\gamma}{\beta})$. It is easy to verify that $\Phi'(\lambda) > 0$ for $\lambda < \frac{1}{t}\ln(\frac{\gamma}{\beta})$, and $\Phi'(\lambda) < 0$ for $\lambda > \frac{1}{t}\ln(\frac{\gamma}{\beta})$. Thus, $\Phi(\lambda)$ is positive at $\lambda = 0$, increasing until $\lambda = \frac{1}{t}\ln(\frac{\gamma}{\beta})$ (if $\gamma > \beta$), and then decreasing. In other words, $v_i(\lambda)$ is single-peaked for $\lambda \geq 0$. $\square$