UNIVERSITY OF CALIFORNIA
RIVERSIDE

Peer-to-Peer Support for Large Scale Interactive Applications

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Yi Hu

September 2012

Dissertation Committee:

Dr. Laxmi N. Bhuyan, Chairperson
Dr. Michalis Faloutsos
Dr. Srikanth Krishnamurthy

The Dissertation of Yi Hu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I own my gratitude to all those people who have made this dissertation possible and because of whom my graduation experience has been one that I will cherish forever.

My deepest gratitude is to my advisor, Dr. Laxmi N. Bhuyan, for his continuous support of my research and study. He guided and inspired me to find and pursue my own ideas. His firm belief in my potential motivated me to overcome all kinds of difficulties in my doctoral study. Without him, this dissertation never would have been done.

I would like to thank my other dissertation committee members, Dr. Michalis Faloutsos and Dr. Srikanth Krishnamurthy for taking their time to help me in this dissertation.

I would also like to thank all my teachers I have had throughout my life. I am where I am today because of them.

Finally, I would like to thank my family, particularly my husband Min Feng, my father Xianming Hu, and my mother Xiaoming Tan, for their unconditional support during these years.

This dissertation is dedicated to my family.

ABSTRACT OF THE DISSERTATION

Peer-to-Peer Support for Large Scale Interactive Applications

by

Yi Hu

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2012
Dr. Laxmi N. Bhuyan, Chairperson

User-interactive applications are evolving in both popularity and scale on the Internet, ranging from simple file-sharing to more demanding interactive applications such as collaborate workspace, and massive multi-player online games (MMOGs). These applications are traditionally implemented by Client/Server architectures, which suffer from significant technical and commercial drawbacks, primarily high-maintenance cost and limited scalability. To overcome these drawbacks, this dissertation presents a Peer-to-Peer (P2P) approach to support large-scale interactive applications.

This dissertation addresses two key design issues for P2P systems to achieve scalability and high performance. The first issue is to provide incentives for users so that P2P systems can aggregate free resources from unreliable users. The second issue is to design consistency maintenance schemes so that P2P systems can provide reliable services to meet the application requirements by using free resources from unreliable users.

For the first issue, this dissertation starts with providing a budget based incentive search service, called BuSIS [106], for efficiently locating service providers in P2P systems. Then, an incentive trading model, called FairTrade [103], is presented for

P2P users to exchange service with each other. Personal currency model is employed in FairTrade to stimulate users to contribute to the P2P community in exchange of desired services. To cope with highly dynamic nature of P2P systems, an enhanced incentive trading model, called CoBank [104], is presented to reduce maintenance overhead at each user and improve robustness against malicious attacks. Cooperative banking strategy is used in CoBank to further distribute the maintenance workload.

For the second issue, this dissertation begins by providing a consistency maintenance framework, called BCoM [105, 102], balancing between consistency strictness, availability and performance for various P2P interactive applications with heterogenous resource constraints. Then, it presents a real-time consistency maintenance P2P system, called PPAct [101], for interactive applications such as MMOGs. View discovery and update dissemination are decoupled in PPAct to mitigate the hot spot problem and ensure consistency maintenance under stringent latency constraints.

Extensive experiments and simulations have been conducted at large scale network scenarios to evaluate the performance of all the works in this dissertation. Results show that comparing with flooding and random walk searches, BuSIS has the lowest search overhead without sacrificing the hit rate. When serving selfish users, flooding and random walk performance degrade dramatically, while BuSIS gracefully keeps the hit rate only with 20% overhead of flooding and 25% of random walk. Applying FairTrade for file-sharing applications, it achieves 100% success rate of download requests without malicious peers, and maintains over 90% success rate even with 50% malicious nodes. The system warms up quickly and does not assume any altruistic service or other kind of help. On average, the system traffic stabilizes before peers issue their second download requests. All these good performances are achieved with extremely low trading overhead, which takes up less than 1% of the total traffic. Compare with

another prominent P2P consistency maintenance scheme SCOPE [55], BCoM outperforms SCOPE with lower discard rates. BCoM achieves a discard rate as low as 5% in most cases while SCOPE has almost 100% discard rate. Evaluating PPAct on two major types of online games: role playing games (RPGs) and first person shooter (FPS) games, the results demonstrate PPAct successfully supports 10000 players in RPGs and 1500 players in FPS games, outperforms SimMud [116] in RPGs and Donnybrook [35] in FPS games by 40% and 30% higher successful update rates respectively.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

User-interactive applications are evolving in both popularity and scale on the Internet, where users exchange information and services with each other in various ways. From traditional file-sharing [9, 4] to recently emerged online social networking [43], collaborate workspace [152], massive multi-player online games (MMOGs) [36, 35], these applications become more demanding for large scalability and high performance. They are traditionally implemented by Client/Server architectures, which suffer from significant technical and commercial drawbacks, primarily high-maintenance cost and limited scalability. To overcome these drawbacks, this dissertation is devoted to providing high performance P2P support for large-scale user-interactive applications.

The term "peer-to-peer" (P2P) refers to a class of systems and applications that exploit resources at the edge of the network (e.g., in homes and offices) to perform a function in a decentralized manner. There is no centralized servers or authorities in P2P systems, each user (also termed as a peer or a node) acts both as a client and a server by contributing its resources and services to the P2P system and getting its tasks done. The nature of P2P systems is well suited for interactive applications, where

users are equally generating and consuming informations or contributing and obtaining resources such as file objects, bandwidth and computation power.

The success of P2P systems relies on the coordination of a large number of peers to contribute their services (e.g., content sharing, online storage, computation) to the P2P community so that their tasks can get completed through the service from the community. Unfortunately, the highly dynamic nature of P2P systems, where users are leaving and joining frequently (i.e., high churn rates), makes it challenging to provide consistent services that satisfy application requirements. Moreover, P2P users are autonomous without any central authorities, they are selfish by nature. Aggregate sufficient resources from such unreliable and selfish users becomes a barrier to the smooth functioning of P2P systems [19, 98].

1.1 Dissertation Overview

This dissertation focuses on two key issues for designing large scale and high performance P2P systems supporting user-interactive applications. The first issue is to provide incentives for users to contribute, and prohibit those who purely take free rides from the P2P systems. The second issue is to provide consistency services satisfying various application requirements by efficiently using resources contributed by users. The major contributions in this dissertation are briefly described below.

1.1.1 Budget-Based Self-Optimized Incentive Search

To realize the function of a P2P system, each user should contribute to the community and help other users finishing their tasks. This reciprocity lays the foundation for each user getting services from the system. This dissertation decomposes

user contribution into two aspects, namely helping others locating service providers, and providing services to others. In this work, an incentive search model is proposed to enable users to help each other in locating service providers. Also, two incentive trading models are proposed for encouraging users to provide services to others.

To help locate service providers means forwarding or replying others search queries according to the search protocol executed in the P2P system. To provide incentives, a search protocol should associate a user's contribution to serving others queries with the quality of search service the user can get from the system. As a result, users with higher contribution get better search service for their own queries. A *budegt-based self-optimized incentive search* protocol (named BuSIS) [106] is proposed in this dissertation to differentiate search service among users based on their contribution. In BuSIS, "credit" is used to represent a user's contribution to serving others queries and the cost of issuing a query search. Each query has a budget prepaid by the issuing peer. The cost of traversing each hop along the search path is deducted from the budget, and the query is discarded when its budget is run out. The cost for a query search accounts for the transmission overhead, and the budget imposes a limit for the overhead a query search will bring to the system. Therefore, the system overhead is balanced with the resource contribution from its users. And the users have incentives to contribute more in order to get better service.

To realize the features of BuSIS, an search performance analytical model is proposed to estimate the expected search quality and the associated cost based on the object popularity and the user reliability. A middleware emulator of BuSIS is developed to evaluate the performance of BuSIS.

1.1.2 Personal Currency Based Incentive Trading Model

After giving incentives in locating service providers through BuSIS, this work proposes an incentive trading model for motivating users in providing services. Reciprocity is used by incentive protocols (e.g., [147, 1, 226, 223, 236, 106, 229]) to enforce service provisions. Generally, there are two types of reciprocity: direct reciprocity (i.e., bilateral trading) and indirect reciprocity (i.e., multilateral trading).

In direct reciprocity, the service provider is instantly rewarded by the receiver, and the reciprocation is synchronized. For example, in BitTorrent systems [1], peers downloading the same file follow a rate-based "tit-for-tat" scheme to exchange file segments. The drawback is that only peers, who are simultaneously interested in each other's services, can trade. Discovering a trading partnership often takes tens of minutes [27] if such a match exists at all. The delay is usually caused by the demand-supply mismatch and unequal peer upload rates. The problem gets worse with churn in P2P networks.

Indirect reciprocity schemes (e.g., [226, 223, 236, 106, 229]) have been proposed to enhance the flexibility and efficiency of direct reciprocity schemes. In indirect reciprocity schemes, the user-provided services are priced in currency units and each user's contribution and consumption of services are measured in the same currency unit. With the currency mechanism, the reciprocation can be asynchronized. The trading partner discovery becomes much simpler and peers are incentivized to contribute even if they are not currently interested in others' services. However, most indirect reciprocity schemes require a central bank or broker to mint currency, maintain each user's account information, resolve disputes, and punish counterfeiters. Such a centralized bank conflicts with the goal of P2P systems.

This work proposes a P2P indirect reciprocity scheme called FairTrade [103], which provides efficient multilateral trading and is resistant to all three major types of attacks in P2P systems (i.e., sybil attacks [69], slander attacks, and whitewashing attacks). FairTrade designs a P2P personal currency model to get rid of any global banks or central authorities required by the global currency model. *To our best knowledge, this is the first work to propose a personal currency based indirect reciprocity model for P2P systems.*

In FairTrade, each peer issues its personal currency as payment for services, which can be spent by its issuer or a third peer who accepted the currency before. A peer accepts its personal currency spent by other peers and provides requested services in return to guarantee the value of its personal currency. The challenge is that the efficiency of a personal currency model relies on the honest behaviors of all participants [25]. However, peers are self-interested and may overissue their personal currencies if they can benefit more. FairTrade handles this challenge by validating a personal currency through its acceptance at other peers instead of its issuing. The validity of a personal currency is measured by how much it will be accepted by other peers. FairTrade introduces *peer credit limit* to measure the maximum amount of personal currency issued by a creditee peer that will be accepted by another peer as a creditor.

A *Bayesian network model* is provided for a peer as a creditor to dynamically set the peer credit limit for a creditee peer by learning the creditee's trading history and estimating the associated risks and profits periodically. Malicious peers are detected and banned through peer credit setting. A P2P overlay simulator is developed to evaluate the performance of FairTrade with and without malicious peers.

1.1.3 Cooperative Banking Based Incentive Trading Model

FairTrade successfully enables incentive trading among peers through personal currency schemes, which achieve the same flexibility for multilateral trading as global currency schemes but in a distributed way. This work aims to further improve the efficiency of incentive trading. In a highly dynamic P2P environment, a uniform global currency is desired due to its low maintenance overhead, however, its centralized nature conflicts the distributed requirement of P2P systems and restricts the scalability. A cooperative banking based incentive trading model, called CoBank [104], is proposed in this dissertation to realize a global currency based incentive trading but in a distributed way.

CoBank supports global currency based trading without requiring any special infrastructure to mint currency or manage accounts, as it employs a cooperative banking strategy. The management of user accounts and transactions is done through the cooperation of peers. To ensure account security, each user account is decomposed into several parts stored at different nodes – account holders and each transaction is performed at a third-party peer – transaction arbitrator. Replication of account data and transaction arbitrator is used to enhance the system robustness.

To enhance the robustness of CoBank in a high-churn P2P environment, an analytical model is proposed for selecting nodes to be account holders and transaction arbitrators. The selection model improves the reliability of a CoBank P2P system while maintaining a sufficient level of scalability. The attack resistance properties of CoBank is examined in the context of three major attacks in P2P indirect reciprocity schemes (i.e., the sybil attacks, slander attacks, and whitewashing attacks). A P2P overlay simulator is developed to evaluate the performance of CoBank.

1.1.4 Maintaining Data Consistency for P2P Interactive Applications

After incentivizing users to contribute resources through previous three works BuSIS, FairTrade, and CoBank, this work begins the second issue in this dissertation that is to provide consistency maintenance for a wide range of user-interactive applications by utilizing the resources contributed from P2P users.

P2P user-interactive applications have various forms, including modifiable storage systems (e.g. OceanStore [121], Publius [212]), mutable content sharing (e.g. P2P WiKi [206]), even interactive ones (e.g. P2P online games [35], P2P Social Networking [43], and P2P collaborative workspace [152]). All these applications involve data replication and require consistency maintenance on data replicas to function properly. These applications have different workload patterns, which may be keep changing. But they all are constrained by the heterogeneous user capacities, and the dynamic P2P settings. Neither sequential consistency [99] nor eventual consistency [199] individually works well in a P2P environment.

Applying sequential consistency leads to prohibitively long synchronization delays due to the large number of peers and the unreliable overlay. Even "deadlock" may occur when a crashed replica node causes other replica nodes to wait forever. Hence, system scalability is restricted due to low data availability resulting from long synchronization delay. At the other extreme, eventual consistency allows replica nodes to concurrently update their local copies, only requiring that all replica copies become identical after a long enough failure-free and update-free interval. Since replica nodes are highly unreliable in P2P systems, the node issuing update may have gone offline by the time update conflicts are detected, leading to unresolvable conflicts. It is infeasible to rely on a long duration without any failure or further updates. As a result, eventual

consistency fails to provide any end-to-end performance guarantee to P2P users.

This work presents a Balanced Consistency Maintenance (BCoM) [105, 102] framework for structured P2P systems to balance between consistency strictness, object availability for updates, and update dissemination latency. BCoM protocol serializes all updates to eliminate the complicated conflict handling in P2P systems. It also allows certain obsolescence in each replica node to reduce the update discard rate of implementing sequential consistency. BCoM limits the extent of temporary inconsistency by developing a sliding window update protocol. The size of the sliding window regulates the number of allowable updates buffered by each replica node. Thus, BCoM provides a measure of consistency guarantee which is specified by an application rather than eventual consistency. BCoM develops an analytical model to set the window size as follows: given an inconsistency bound, the window size is set to minimize the update discard rate while ensuring the expected delay is no worse than the baseline by a small given threshold. Two enhancement schemes are presented to improve the fast recovery of node failure in BCoM and to reduce the impacts of bottleneck nodes. The P2PSim [12] simulation tool is customized to evaluate the performance of BCoM with a real data case study.

1.1.5 Real-time Consistency Maintenance

After presenting a consistency maintenance framework BCoM, this work proposes a P2P system, called PPAct [101], supporting real-time consistency maintenance for large-scale interactive applications. Massive Multi-player Online Game is used as an example application to illustrate the system, while it can be applied directly to all P2P interactive applications.

A core challenge of providing real-time consistency maintenance in P2P systems

is to ensure update delivery under a stringent latency constraint only using the uplink bandwidth from peers. In a MMOG, the total number of updates sent by players is quadratic to the total number of players. But the total supply of uplink bandwidth used for update delivery increases only linearly with the number of players. The bandwidth shortage limits the scalability of P2P systems.

PPAct adopts the Area-of-Interest (AOI) filtering, which is proposed in prior works [36, 116] to reduce the number of updates sent to each player. The game map is partitioned into regions and each player receives updates in surrounding regions. These regions are termed as *view regions*. Each player subscribes to its view regions in a real-time fashion. However, AOI filtering only alleviates the bandwidth shortage but does not solve it fundamentally. With imbalanced workload, AOI filtering has a critical "hot spot" problem, where the players responsible for sending updates about popular regions still lack of bandwidth and fail to deliver on time.

PPAct solves AOI's "hot spot" problems through dynamically balancing the workload of each region in a distributed way. In PPAct, the roles of view discovery are separated from consistency maintenance by assigning players as "region hosts" and "object holders". Region hosts are in charge of tracking objects and players within a particular region, and object holders are in charge of sending updates about a particular object to interested players. Lookup queries for view discovery are processed by region hosts, while consistency maintenance of objects are taken by object holders. This separation distributes the workload and simplifies the lookup procedure and update delivery. Another key idea in PPAct is that peers contribute spare bandwidth in a fully distributed way to forwarding updates about objects they are interested in. Thus popular objects for which demands are higher will have more peers forwarding updates for them. A node selection model is presented to select region hosts and object holders

with capability and reliability considerations. A P2P network simulator is developed to evaluate PPAct on two major types of online games: role playing games (RPGs) and first person shooter (FPS) games.

1.2 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 reviews literature work. Chapter 3 presents the budegt-based self-optimized incentive search model to provide incentive for service location in P2P systems. Chapter 4 describes the personal currency based incentive trading model to provide incentives for service provision in P2P systems. Chapter 5 presents the cooperative banking based incentive trading model to enhance the efficiency of incentive-aware service provision. In chapter 6, a data consistency maintenance framework is presented to address the generalized consistency requirements from a wide range of P2P interactive applications. Chapter 7 presents a real-time consistency maintenance system for P2P interactive applications with stringent latency constraints. The conclusions of the dissertation are summarized in Chapter 8.

Chapter 2

Related Work

2.1 Incentive Models in P2P Systems

2.1.1 Incentive Search Models in P2P Systems

There is few literature works on incentive-aware search model for P2P systems. The BuSIS proposed in this thesis is one of the pioneer works in this area. The authors of [131] address the incentive query propagation in unstructured P2P systems, but they assume every node is identical and behave the same. This assumption is a fatal conflict to real P2P systems, where nodes are highly heterogeneous and autonomous, each behaves to maximize its own interests. While, BuSIS handles node heterogeneity by tailoring each search query based on the issuing node capability and the queried object popularity.

The predominant search model for unstructured P2P systems is flooding, like in Gnutella systems [8], where a search query is broadcast and rebroadcast until the object is found, and a reply message is sent back to the requester following each inverse search path. Or the query is dropped when its time-to-live (TTL) counter reaches zero. Flooding has many sound features fitting the dynamic P2P environments, such as its in-

dependence of the underlying infrastructure, tolerance to nodes failures, and adaptation to the dynamic network topology. However, the excessive overhead of flooding restricts the scalability of P2P systems. Many literature works have been devoted to improve the search efficiency of flooding from three aspects: 1) by changing the search behavior and limiting the flooding degree, like random walker [141] [85] reduces traffic overhead from exponential increase to linear increase; 2) by exploring the cache information like DiCAS [213] or the congestion information like Congestion-aware Search [123] to restrict the search scope and cut down the overhead; 3) by matching the physical network topology with the overlay topology like LATM [139] to trim the redundant traffic in the underlying physical links. Random walk gains popularity by its simplicity and low overhead, while this comes at the cost of easy convergence (rare objects in low connectivity nodes are harder to be found), low reliability (single node crash leads to search failure) and high delay (proportional to the search length) due to its linear search. Hence, hybrid methods of flooding and random walk (e.g. Percolation Search [176] and BubbleStorm [197]) are proposed to take advantages of strength of each method. Also an adaptive random walk search based on object popularity is presented in [37]. BuSIS adopts the simple and lightweight random walk search as basis, the hybrid of one-step flooding is an option here, because of the in-negligible overhead of exchanging and updating the sharing folders between neighbors. On top of this, BuSIS provides differentiated search as an incentive to selfish users, which are not concerned by the above schemes and makes them vulnerable to selfish user behaviors.

2.1.2 General Incentive Trading Models in P2P Systems

A number of papers from the early to mid 2000s laid the foundation for providing incentive trading in P2P systems (e.g., [89, 97, 112, 137]). The authors of [89]

formally analyzed the equilibria of user strategies in P2P file-sharing systems for different kinds of payment mechanisms. A utility function is introduced in [97] to measure user contributions and an auditing scheme is discussed to maintain the integrity of the values of the utility function in the case of user cheating. The authors of [112] gave each peer a EigenTrust score to indicate how many rewards the peer is entitled to. The authors of [137] addressed the asymmetric download and upload bandwidth and proposed using a token based accounting scheme to provide asymmetric incentives to peers. All of these works made assumptions on peer's availability and assumed a fixed topology, while, the FairTrade model presented in this thesis learns dynamically from the changing P2P environments to set credit limit for peers and adapts accordingly.

Recently, a few research works [147, 125, 140] investigated the linkage between direct and indirect reciprocity and proposed incentive schemes residing in between of the two extremes. The authors of [147] gave a theoretical efficiency bound comparing the two reciprocities. The authors of [125, 140] present incentive schemes extending direct reciprocity towards indirect reciprocity for better efficiency and flexibility. Both have similar core techniques: an indirect trading is decomposed to hop-by-hop contribution transfer and direct trading. The difference is that a social network [185, 224] is given in [140] to perform the contribution transfer, while in [125] each peer has to discover its local view of the contribution network to find a valid transfer path. The indirect trading in [140, 125] is restricted by an intermediate bottleneck node in the contribution transfer path, while the personal currency of FairTrade fundamentally solves the limitation imposed by intermediate nodes. Each peer can use personal currency either issued by itself or by others to trade directly. Any two peers can be trading partners without any topology or routing constraints.

The credit setting is an important account management tool in financial worlds.

Several statistical and operational research methods have been applied to credit scoring, as surveyed in [202].These economic tools are too complicated to implement in P2P systems. Instead, P2P schemes use one or two simple calculation formulas to set credit: for example, PledgeRoute [125] uses a monotonically sublinear growing formula and NABT[140] uses an additive increase multiplicative decrease formula. The most critical drawback of these simple P2P credit setting formulas is that they only consider the trading history of the customer peer with an uploader but not with others; as a result, their credit settings do not consider the system-wide contributions and takings, which FairTrade credit setting model takes into consideration. Thus, peers in FairTrade detect malicious attacks more efficiently with low the maintenance overhead for credit setting.

2.1.3 Indirect Reciprocity Models in P2P Systems

Global currency is popularly used by P2P indirect reciprocity models, where a user earns currency units when providing service and spends when acquiring service. Users' savings could be tracked by using virtual currency and a central bank (e.g.,[182, 27]) or some form of digital cash and a broker or brokers (e.g., [226, 223]). Economic theory shows that global currency systems are highly efficient [207]. However, the central bank/broker conflicts with the distributed P2P system. The CoBank model presented in this thesis adopts the global currency approach but gets rid of any central mechanism by fully distributing the management workload of accounts and transactions to cooperative peers. Karma [211] system provides distributed account management, however, it purely uses replication to store account information thus subjects to sybil attacks and other account abuses. To the contrary, CoBank separates the transaction arbitrators from the account holders, and explores the cooperation among different account holders to combat all major attacks. CoBank also uses replication to further enhance the security.

Pairwise currency is another popular approach to enable indirect reciprocity for P2P users (e.g., [192, 125]), where each user maintains a distinct currency/credit account for every other peer. The robustness and efficiency of pairwise currency schemes reside in between those of bilateral trading (i.e., bartering) schemes and global currency schemes. It allows asynchronous bilateral trading among networked nodes and provides distributed account maintenance. A critical problem of pairwise currency is that each user needs to maintain $O(N)$ number of distinct accounts, where N is the total number of nodes in the network. Thus, there are total $O(N^2)$ accounts maintained in the system. Such overhead severely reduces the scalability of P2P systems. Moreover, for each transaction the buyer has to find a valid currency/credit transfer path in the contribution network as in PledgeRoute [125]. In the worst case, finding such a path may have $O(N)$ communication overhead because of node churn or insufficient currency at the intermediate nodes. In CoBank each user only has $O(1)$ storage overhead and each transaction only has $O(\log N)$ communication overhead. CoBank also achieves distributed management through cooperative banking.

Social network is introduced to reduce the account maintenance overhead of pairwise currency approaches (e.g., [140, 162]), where each user only maintains accounts for social peers. In [140], peers only maintain accounts for directly connected peers (i.e., one-hop neighbors) in the social network. Trading with other peers has to be done through currency/credit transfer along a valid path in the social network. In [162], users maintain accounts for indirect connected social peers and calculate social distance for each social peer. Their service provision is in reverse to the social distance. A critical limitation of such schemes is that they rely on a given social network. Such social network usually is not available, since users may not willing to disclose their social relationships while participating in a P2P system. CoBank reduces the account

management overhead and does not have any underlying restriction.

2.2 Consistency Maintenance in P2P Systems

2.2.1 Consistency Maintenance in Structured P2P Systems

In structured P2P systems, strong consistency is provided by organizing replica nodes to an auxiliary structure on top of the overlay for update propagation. Examples include the tree structure in SCOPE [55], the two-tiered structure in OceanStore [121], and a hybrid of tree and two-tiered structure in [136]. The tree construction algorithms in SCOPE [55] and in [136] build a tree by recursively partitioning the identifier space and selecting a representative node as a tree node for each partition. Only leaf nodes store object copies, all the intermediate nodes only store information of the tree structure in their sub-space. Nodes who may not be interested in the object are in the object's update dissemination tree, which adds unnecessary overhead of maintaining the tree from node failures. To the contrary, the BCoM model presented in this thesis constructs the dissemination tree dDT by only involving replica nodes who are interested in the object, which greatly reduce the overhead of maintenance and update propagation. BCoM also efficiently builds the dissemination tree dDT to make it balanced and robust under the node churn.

2.2.2 Consistency Maintenance in Unstructured P2P Systems

In unstructured P2P systems, mainly two types of bounded consistency are provided: (1) probabilistic bounded consistency: rumor spreading [63] and replica chain [220] are used to ensure a certain probability of an update being received. The probability is tuned by adjusting the redundancy degree in propagating an update to balance the

communication overhead with the consistency strictness. (2) time-bounded consistency: TTL guided push and/or pull methods are used (e.g., [138] [194]) to indicate a valid period for a replica copy. When the period expires, the replica node checks the validity of the replica copy with the source to serve the following read requests. The problems of these techniques are (1) node-level consistency is not ensured by probabilistic bounded consistency, and (2) time-bounded consistency sets a uniform TTL timer for all nodes. In the situation where nodes have various update frequencies, it is impossible to set a TTL timer that works for all nodes. BCoM avoids both drawbacks by using a sliding window update protocol, which directly limits the inconsistency by the number of buffered updates and ensures the bounded consistency for each replica node.

2.2.3 Tunable Consistency Models in P2P Systems

Previous works [117][230] have proposed continuous models for consistency maintenance, which have been extended by a composable consistency model in [191] for P2P applications. The core technique for maintaining consistency used in [191] is a hybrid of push and pull methods, which are also used to provide application tailored cache consistency in [239] [138]. Although each node can specify its consistency requirement, the model in [191] makes each node perform the strongest consistency maintenance from all its descendant nodes in the overlay replica hierarchy. Thus, the overhead of maintaining consistency at a node is not reduced even it only requires a weak consistency as long as one of its descendant nodes requires a strong consistency. An analytical model for adaptive update window protocol is presented in [235], where the window specifies the number of uncommitted updates in each replica node's buffer. The information of each node's update rate and propagation latency are required to optimize the window size in [235]. Such optimization is unrealistic for P2P systems due to their required global

information. To the contrary, every BCoM node has a fair amount of consistency maintenance overhead because of the uniform buffer size and node degree in dDT. Moreover, BCoM provides incentives for nodes to contribute more bandwidth to update dissemination, as we promote faster nodes closer to the root and they will receive updates sooner. The window size optimization model in BCoM only requires limited information that can be obtained in a fully distributed way.

2.2.4 P2P Managed Online Game Systems

In the literature, two major techniques are proposed to reduce bandwidth demand for update delivery. One is Area-of-Interest (AOI) filtering [36, 116], where updates of an object is only sent to the players within the same region as the object. This method works well when players have limited proximity to each other, but does not help when players are clustered in a few places, which is unavoidable due to the power law distribution of the player population density [160]. As a result, the demand in such *hot regions* still grows quadratically, and AOI does not fundamentally solve the bandwidth shortage problem. The other technique is proposed in the system Donnybrook [35], which limits each player to receive only a constant number of real-time updates. The demand of uplink bandwidth for sending real-time updates is then linearly proportional to the number of players. Since every player still has to be aware of the status of all others, each Donnybrook player is required to broadcast less frequent guidance messages. Therefore, their total demand of uplink bandwidth remains asymptotically quadratic to the total number of players.

The PPAct system presented in this thesis follows the AOI filtering but solves its "hot spot" problem by having players contribute their spare bandwidth to update delivery in a distributed way. We also take advantage of players' movement patterns to

build a 2-D DHT for reducing the lookup overhead and delay. Thus, more bandwidth is saved for sending out update on time.

For better workload distribution, PPAct system decouples the view discovery and update delivery similar to the Colyseus system [36], which decoupled object discovery and replica synchronization. But Colyseus system [36] assumed objects were properly placed with minimized interactive latency without any specific methods. PPAct system devises efficient schemes for handling object placement, peer selection, and failure recovery.

2.2.5 Load Balance on P2P Networks

Data replication and load redistribution are two major solutions to the imbalance workload problem in P2P networks. The workload imbalance is due to the skewed data popularity and heterogeneous peer capability. Data replication alleviates the overloaded nodes by providing extra targets for the incoming requests [235, 191, 90, 200]. Data migration requires actual data transfer between nodes to balance workload, which has two major forms: data item exchange and node migration. Optimal load balance needs use both item exchange between neighbor nodes and global node migration [78]. Some P2P networks use topology adjustment to assist load balance. A node in the Mercury system [34] uses sampling to estimate other nodes' workload and adjusts its long link connections for load balance. The authors in [181] propose that each node tunes its routing table size to balance the query workload and avoid congestion. These methods focused on static traffic, PPAct addresses load balance for dynamic traffic and avoids data shipment or node migration.

To balance the workload for update delivery, Donnybrook [35] also has each player advertise its spare bandwidth to help forward updates. However, in Donnybrook

[35] each player has to broadcast its spare bandwidth, which requires the global knowledge of all players and strictly limits the system scalability.

2.2.6 P2P Support for Range Queries

Distributed Hash Table (DHT) provides a scalable and robust substrate to build structured P2P systems. However, DHT only supports exact-match queries because the uniform hashing of DHT destroys the data locality required by range query processing. Existing P2P index techniques to support range queries fall into two categories: (1) DHT-preserved indexing, which maintains the original DHTs and builds an overlay index-structure on top of DHTs (e.g., Prefix Hash Trie (PHT) [53], multi-dimensional Lightweight Hash Tree (mLIGHT)[195], Distributed Segment Tree (DST) [240] and Range Search Tree (RST) [80]); (2) DHT-modified indexing, which modifies the internal structure of DHTs and develops certain locality-preserved overlay (e.g., multi-ring structure in Mercury [34], SkipIndex [234] incorporate space kd-tree index into skip-graph). However, all these methods are infeasible to support real-time range queries, where the answer is time related and the object attribute is changing dynamically. Every time the change of object's attribute value leads to changes on PHT or object movement from one node to another. This results in cascading changes and requires re-running load balance algorithms. PPAct successfully enhances the DHT structure to support real-time range queries by properly partitioning regions as unit of range index.

Chapter 3

Budget-Based Self-Optimized Incentive Search

3.1 Budget-Based Self-Optimized Incentive Search Model

BuSIS consists of four components: Search Performance Estimation (SPE), Budget Assignment (BA), Query Forwarding (QF) and Parameter Maintenance (PM). BuSIS locates between application layer and transport layer and makes all its optimizations transparent to end uses. It can integrate with any transport layer protocol, for example, our emulator builds BuSIS on top of TCP to run in the Internet. To request a search, the requester peer first runs the SPE module to estimate search performance for a certain set of operation parameters (total TTL, and the number of simultaneous walkers). Then the requester runs the BA module to choose the most beneficial pair of operation parameters, and assign budget on the query walkers. The QF module is run by a peer to send out a query after assigning budget and to relay others' query when the query does not hit locally. A peer runs the PM module to update the input parameters of the PE, BA and QF modules. The update is performed whenever the peer sends out

a query, receives a query and receives a query feedback. The feedback message is sent along the reverse search path after a query hits.

BuSIS models the unstructured P2P file sharing system as a *competitive market*[207], where each peer acts as a consumer or a producer. The major services provided in the P2P file sharing market include object search and file transfer, and BuSIS focuses on object search service. A peer takes the role of a consumer when initiating a search query, and it pays the search service by associating an amount of budget on the query. When a peer receives other's query, it plays as a search service producer; it processes the query and deducts its charges from the query budget. If it holds the query object, it replies to the requester by a query feedback message, otherwise, it relays the query to another neighbor for further search. In BuSIS, each peer charges the same for both query hit and query forwarding in terms of providing search service. After receiving the query feedback, the file host charges a larger amount of credits for the real file transfer. The credit is the currency in our P2P market and in this paper, we assume no fraud in currency implementation. The techniques for secure currency implementation are discussed in [76] [241] [226].

Initially a peer enters the market without any credit, it gets the *participation payment* from the system for each on-line time-slot. This is regardless of whether this peer shares objects or forwards queries. After establishing connections with neighbor peers, it receives others' queries, processes and charges them. If the query matches the local sharing object, it replies a feedback message to the query requester. On receiving a feedback message, each relay peer along the reverse search path runs the PM module to update the related input parameters. If the peer does not have the object, it runs the QF module to forward the query to another neighbor. In both cases this peer charges the query the same price. In case the query lacks budget, if the peer holds the object it

still charges and sends back the feedback message, indicating the amount the requester needs to pay later. If the peer does not have the object, it stops forwarding and drops the query with insufficient budget.

To search an object, the peer first runs the SPE module to estimableate the expected search performances for all sets of feasible operation parameters, which are then used by the BA module to choose the most beneficial one and set the query budget to be the associated search cost. To issue a search query, the requester peer should wait until it earns enough credits to prepay the budget determined by the BA module. After accumulating enough credits, the peer runs probabilistic forwarding in the QF module to send out each walker of the query search and deducts the budget from its credits account. If the query hits and the requester successfully receives the feedback message, it directly contacts the file host to perform file transfer.

Figure 3.1 shows the basic operations and data flowchart of the BuSIS protocol. The equations cited in the flowchart are used to compute various cost functions based on which the decisions are made. For better readability, we describe the protocol operations in this section and derive the equations in Section 3.2. The detailed derivations may be skipped without affecting the flow of the paper. In the following, we elaborate the four modules in detail.

3.1.1 Search Performance Estimation (SPE)

Given certain operation parameters, the SPE module estimates the expected search performance. BuSIS is built on top of the random walk search, where the requester sends out several walkers. Each walker is assigned a TTL and randomly searches along a path until a hit occurs or its TTL reaches zero. SPE module uses the total TTL

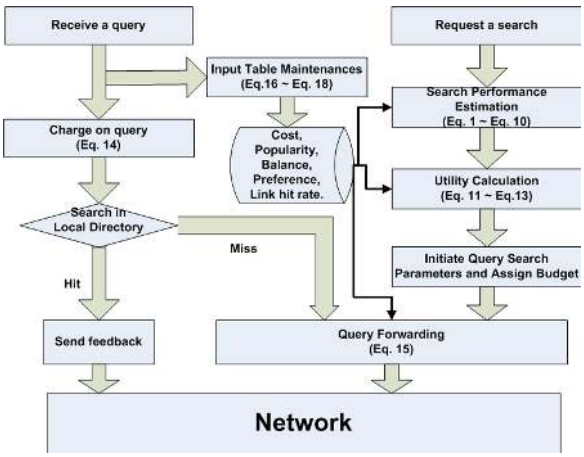

Figure 3.1: BuSIS Flowchart

T_q and the number of simultaneous walkers x_q as the operation parameters to estimate the expected hit rate $prob_q$, hit delay del_q and search reliability R_q, assuming each walker equally shares the total TTL. For better readability, we put the detail derivation steps in Section 3.2.1.

3.1.2 Budget Assignment (BA)

Instead of setting uniform TTL, BuSIS uses the budget to control the number of search hops for each walker. The requester peers prepays an amount of budget for a query search, which is the associated search cost for an expected search performance. Clearly, low budget limits the search scope and can result in a low hit probability, while surplus budget wastes resources and causes unnecessary traffic overhead. The total budget per query can also be split among the multiple simultaneous walkers to shorten the search delay, while the number of simultaneous walkers cannot exceed the node connection degree. In case of query hit, the remaining budget is saved and returned to the requester. But for multiple walkers, even if one of them hits, the other ones will still use up their budget. The BA module determines the budget amount and the

simultaneous walkers number with the aim to maximize the search requester's interests, represented by the utility function. The cost and utility functions to determine the budget assignment are derived in Section 3.2.2 for better readability.

A selfish peer is modeled as a utility maximizer, making all choices toward maximizing its search utility. Since the specific form of utility function is essentially unknown, we aim to formulate it in a simple form and properly reflects the impacts of relevant factors.

The requester i scans all feasible pairs of total TTL T and the number of walkers x, calculates the resulting utility from each pair, and chooses the pair (T, x) with the maximum utility as Equation 3.12 in Section 3.2.2. The utility goes up with better performance and gets lowered by larger search cost, and the maximum utility balances search performance with the cost, which automatically constrains the total search overhead. This makes bound the total TTL explicitly unnecessary. But without any constraints, the peer will initialize queries once they earn a little credits. The query budget is subject to the amount of credits this peer currently has, even under the utility function control, issuing queries with insufficient credits leads to poor search performance and large overhead. Therefore, the utility function requires that the estimated hit probability should be no worse than the acceptable hit probability threshold P_{thre}. Furthermore, to adapt to time sensitive applications, the estimated hit delay should be no worse than the delay threshold D_{thre}.

After selecting the pair (T, x) with maximal utility, the requester i waits to earn enough credits to prepay the associated cost $C_i(T, x)$ as the budget, then sends out x simultaneous walkers with equal share of budget.

3.1.3 Query Forwarding (QF)

The QF is run by a peer for two cases, one is for forwarding other's query because no local objects match, and the other is for sending out the query initiated by itself. The forwarder adopts the probabilistic forwarding assisted by the knowledge of the object popularity, outgoing link quality and network condition. The goal of QF is to increase the query hit probability for future search under its budget constraint.

First, if the query is other's, the local peer i checks the remaining budget of the query and computes its search price X^i_{srch}. The query is dropped if the remaining budget cannot afford X^i_{srch}. The search price is determined by the link quality, the more neighbor connections the more choices it has to forward the query; the shorter queuing delay the faster the query will move forward and the lower risk being dropped due to buffer overflow. Better link quality deserves higher charge. The search price calculation is given in Section 3.2.3. We allow each peer to locally compute its price, assuming the peers are honest. To prevent fraud in price calculation, some security techniques can be applied, like having the system administrator conduct random anonymous sampling to examine the price.

The peer i takes inputs as the object popularity pop^x_i of the query object x, and the outgoing link hit rate $hit_{i,k}$ to calculate Pr^x_k, the forward probability to neighbor k for searching object x. The calculation of Pr^x_k is given in Section 3.2.3.

When the object holder is unknown, probabilistic forwarding gives each outgoing link some chances to find the object. For a popular object, an existing good path is preferred, so the forward probability is biased on high hit rate link. For a rarer object, uniform random forwarding is chosen to explore potential good path, so the forward probability equally distributes to each link. Since biased on high hit rate links con-

verges the search path to those "hot peers", which have high connectivity and can easily be visited, and restricts the search scope. This will make some low connectivity peers rarely be visited.

Finally, peer i charges this hop cost X^i_{srch} from the query budget, updates the query remaining budget, and forwards the query.

3.1.4 Parameter Maintenance (PM)

The PM module maintains five input parameters used by the SPE, BA, QF modules, which are per hop search cost, object popularity, link hit rate, credit account balance, and performance threshold. The first three are periodically updated. We consider a discrete time domain, where time is slotted and one time slot equals a update duration. The estimated value of these three parameters are weighted averages of the previous estimation and the most recent estimation. We use α to denote the weight of the previous estimation, where $0 < \alpha < 1$. Whenever a peer receives a query, or a feedback message, it examines the packet header and updates the most recent estimation of the related parameters. No extra communication is required to perform the parameter maintenance. The exact update equations are given in Section 3.2.4.

3.2 BuSIS Analytical Model

3.2.1 SPE Derivations

Following some analytical expressions in [37], this section gives detail derivations of the search performance estimation made in the SPE module.

Hit rate: Given a feasible operation parameter pair: total number T_q of hops and number of simultaneous walkers x_q, we use a uniform random distribution model.

Assuming p is the probability that a peer holds the target object (p is the expected popularity of the object). Equation 3.1 is the expression of the expected hit probability $prob_q$.

$$prob_q = 1 - (1-p)^{T_q} \tag{3.1}$$

Hit Delay: We measure the expected hit delay del_q by the number of hops before the walker terminates, which we denote as D, and assume that ρ is the average per hop delay. $E[D]$ is the expected value of D, and for a single random walker, the termination probability at each hop is defined in Equation 3.2, since the walker terminates either by a query hit or exhausting T_q hops. Thus the expected hit delay for a single random walker is defined in Equation 3.4, and $E[D]$ is defined in Equation 3.3.

$$P(D=j) = \begin{cases} p*(1-p)^{j-1} & 1 \le j < T_q \\ (1-p)^{j-1} & j = T_q \end{cases} \tag{3.2}$$

$$E[D] = \Sigma_{j=1}^{T_q} \;\; j * P(D=j) \tag{3.3}$$

$$del_q = E[D] * \rho \tag{3.4}$$

In the case of x_q simultaneous random walkers, and each with $\frac{T_q}{x_q}$ hops for searching, the expected delay of each walker $E[D_x]$ is defined in Equation 3.6. And the expected hit delay for simultaneous x walkers is defined in Equation 3.7.

$$P(D_x=j) = \begin{cases} (1-(1-p)^x)*(1-p)^{x(j-1)} & 1 \le j < \frac{T_q}{x_q} \\ (1-p)^{x(j-1)} & j = \frac{T_q}{x_q} \end{cases} \tag{3.5}$$

$$E[D_x] = \Sigma_{j=1}^{\frac{T_q}{x_q}} \;\; j * P(D_x=j) \tag{3.6}$$

$$del_q = E[D_x] * \rho \tag{3.7}$$

Search Reliability: Since the query hit reply propagates back to the requester through the reverse search path, any relay peer on the path may disconnect

after forwarding the query but before the reply is back. So we model the reliability for a query hit path, which reflects the possibility that the query hit result may never reach the requester and should be considered in the requester's utility function. Let t_{on} be the average peer online time slots, and $E[D_x]$ be the number of hops the query hit away from the requester. t_{on} can be set locally by observing neighboring peers average online time. For easy notation, set y equal to $E[D_x]$ and relay peers are ordered as $p_1, p_2 \dots p_y$ based on their hops from the requester. We define the holding period for each relay peer p_j $(1 \leq j \leq y)$ in Equation 3.8 as the number of time slots it should wait for the hit reply after it forwards the query.

$$P_{hold}^{j} = 2 * (y - j) \tag{3.8}$$

Then the probability for p_j $(1 \leq j \leq y)$ keeping online during its holding period is given in Equation 3.9.

$$p_{on}^{j} = (1 - \frac{1}{t_{on}})^{P_{hold}^{j}} \tag{3.9}$$

Thus, the expected search reliability R_q associated with (T_q, x_q) is the probability that all relay peers remain online during the whole query hit reply period, which is given in Equation 3.10.

$$R_q = \prod_{j=1}^{y} p_{on}^{j} \tag{3.10}$$

3.2.2 BA Calculations

Search cost: Let $C_q^i(T_q, x_q)$ denote expected cost for peer i to conduct a search with T_q total hops and x_q simultaneous walkers. Let C_{hop}^i be the estimated per hop cost at peer i. The expected total cost $C_q^i(T_q, x_q)$ pays the estimated hops before each independent walker terminates and is defined in Equation 3.11.

$$C_q^i(T_q, x_q) = x * C_{hop}^i * \{\Sigma_{j=1}^{\frac{T_q}{x_q}} \quad j * P(D = j)\} \tag{3.11}$$

Search requester's utility: The utility function, defined in Equation 3.13, takes the total hop counts T_q and the number of simultaneous walkers x_q as input and follows the derivations in Section 3.2.1 to get the expected hit probability $prob_q$, hit delay del_q, search reliability R_q and search cost $C_q^i(T_q, x_q)$.

$$(T, x) = arg \ \max \ U_i(T_q, x_q) \ s.t. \ prob_q \geq P_{thre}, \ del_q \leq D_{thre} \tag{3.12}$$

$$U_i(T_q, x_q) = \frac{prob_q * R_q}{C_q^i(T_q, x_q)} \tag{3.13}$$

In Equation 3.13, a weight can be added to the estimated hit probability as an coefficient to scale the impacts of hit probability and search reliability. In this work, we treat them equally.

3.2.3 QF Calculations

Search price calculation: The search price X_{srch}^i of peer i is defined in Equation 3.14, where $\mathcal{N}_i$ is the connection degree of peer i, L_i and $\mathcal{Q}_i$ are the average link and queuing delay of peer i, which consist of one hop delay. ω_{hpd} is the weights on one hop delay, which tunes the connection degree and per hop delay to be comparable. In our experiment, we set ω_{hpd} based on the relationship between node average connection degree and estimated per hop delay.

$$X_{srch}^i = \mathcal{N}_i + \frac{\omega_{hpd}}{(L_i + \mathcal{Q}_i)} \tag{3.14}$$

Forward probability of an outgoing link: The forward probability to neighbor k for searching object x: Pr_k^x is defined in Equation 3.15, where S_i denotes the neighbor set of peer i, $hit_{i,k}$ denotes the hit rate of outgoing link (i, k), which is the ratio between the number of query hit from this link and the total number of queries

forwarded through this link. pop_i^x is the query object popularity observed by peer i.

$$Pr_k^x = \frac{hit_{i,k}^{pop_i^x}}{\sum_{q \in S_i} hit_{i,q}{}^{pop_i^x}} \tag{3.15}$$

3.2.4 PE Calculations

Parameter Update:

1. Per Hop Cost

   Let $C_i^{hop}(t)$ denote the estimation in the t^{th} time slot, α denote the weight of previous estimation, $C_{total}^i(t)$ denote the total search cost the peer i counts during the t^{th} time slot, and $N_{hop}^i(t)$ denote the total search hops paid by $C_{total}^i(t)$. $C_{total}^i(t)$ and $N_{hop}^i(t)$ are computed by examining the query packets passing through peer i, and the query results replying to peer i during the t^{th} time slot. The updating is performed as in Equation 3.16.

   $$C_{hop}^i(t) = \alpha * C_{hop}^i(t-1) + (1-\alpha) * \frac{C_{total}^i(t)}{N_{hop}^i(t)} \tag{3.16}$$

2. Object Popularity

   The estimated popularity of the object x is derived from the average hit rate. Let $q_i^x(t)$ denote the estimated popularity in the t^{th} time slot, and $pop_i^x(t)$ denote the weighted average popularity until the t^{th} time slot. The estimated popularity is updated as Equation 3.17.

   $$pop_i^x(t) = \alpha * pop_i^x(t-1) + (1-\alpha) * q_i^x(t) \tag{3.17}$$

3. Link Hit Rate

   The estimated hit rate of an outgoing link is the ratio of total number of queries forwarded through that link to total number of feedback messages received through that link, the final result is averaged over time. For each neighbor j, let $hit_{i,j}(t)$

denote the weighted average hit rate examined by peer i up to the t^{th} time slot for outgoing link (i, j), $h_{i,j}(t)$ denote the estimated hit rate in the t^{th} time slot. The link hit rate is updated as Equation 3.18.

$$hit_{i,j}(t) = \alpha * hit_{i,j}(t-1) + (1-\alpha) * h_{i,j}(t) \tag{3.18}$$

4. Credit Account Balance

   C_i denotes the total available credits at peer i, which is updated after each time peer i issues a search, forwards and replies others' queries. For security reason, some authentication and forge-proof mechanism must be used to manage the peer credit account. In our protocol, we adopt the simple receipt method and assume that each peer is honest in maintaining the credit account.

5. Performance Threshold

   Performance threshold includes the thresholds of hit probability P_{thre} and hit delayD_{thre}. These constants are set once at initialization to prune out infeasible operation parameters.

3.3 Performance Evaluations

In this section, we develop a custom emulator to evaluate the performance of BuSIS, and compare with random walk and flooding search techniques. We introduce our experimental methodology in Section 3.3.1 and explain the network figuration in Section 3.3.2. Then, in Section 3.3.3, we analyze the experimental results, elaborate the implications of the figures and discuss some important performance issues.

3.3.1 Experimental Methodology

It is infeasible to configure a large number of peers in Gnutella network with BuSIS to evaluate the performance. Thus, we develop a BuSIS-emulator middleware, which can be deployed in a large distributed system to form a BuSIS-based unstructured P2P network. Our BuSIS-emulator creates multiple independent processes; each process representing a single peer and implementing BuSIS middleware on top of TCP/IP protocol to run in the Internet. A peer is emulated for joining and leaving the network, establishing and maintaining connections with neighbor peers, generating queries, processing and replying others' queries, and updating sharing directories to simulate file transfers. Current emulations are based on deployment in the LAN in our computer science department to simulate the performance with 1000 nodes. Our emulations aim to verify:

1. the adaptivity and self-optimization capability of BuSIS in the face of changing object popularity;

2. the effectiveness of BuSIS under selfish and selfless user behavior;

3. the positive impact of BuSIS on smoothing the query injection in case of traffic burst;

4. the effectiveness of BuSIS for heterogeneous peers topologies.

We design different emulation scenarios to fulfill the above four objectives, and the results are presented in the following sections.

BuSIS strives for both better user-perceived search quality and system efficiency, so we focus on two metrics: the *hit rate* and the *overhead-per-query*. Hit rate is defined as the number of query hits over the total number of issued queries. And the

Table 3.1: Summary of parameters setup

Symbol	Definition	Value
P_{thre}	hit probability threshold	0.5
D_{thre}	hit delay threshold	10s
ω_{hpd}	weight of per hop delay in search price calculation	0.1
α	weight of previous estimation for periodic updates	0.8

overhead-per-query is defined as the total search overhead (total number of hops queries traversed) divided by the total number of issued queries.

3.3.2 Network Configuration

There are total five parameters need to be set up, which are summarized at Table.3.1. Their setup values is based on their functionalities we explained when introduced them.

When a new peer joins, it first tries its cache, which contains addresses of previous neighbor peers and peers who replied its queries before, to establish neighbor connections. If those cached peers are not active, the new peer asks a well known node (landmark) for a list of candidate neighbors. We bound the maximum connection degree of each node to x_{max}. Then the landmark node sends back a list of x_{max} recent active peers. The new node connects to neighbors from this list. When a peer's connection degree becomes smaller than half of x_{max}, it re-sends request message to the landmark node for more candidate neighbors. In emulation we set the maximum node degree to 5, based on the average Gnutella node degree of 3 to 5.

P2P networks are highly dynamic with peers that go on-line and off-line frequently. It has been shown that 20% of the P2P connections last no more than 1 minute, and around 60% of peers keep on-line no more than 10 minutes each time after they enter the system [179]. We emulate the dynamics of P2P networks by assigning each peer

a lifetime, which is the time period the peer will be online in the system. The lifetime follows the distribution observed in [174], with the mean value to be 10 minutes [179]. During each second, a number of peers leave the network with there lifetime reduced to zero. We let the same number of new peers enter the network to keep the network size as 5000 peers. The query generation by a peer follows a poisson process with inter generation time of 5 seconds. The number of replicas of any object ranges from 10 to 200, all of which are randomly spread in the network. Each peer queries objects which are locally unavailable with the query object popularity simulated as Zipf distribution [41].

For TTL setting, BuSIS initiates a query search by determining the number of simultaneous walkers and allocating a budget to each walker. The actual TTL of each walker is determined by the assigned budget and the charges of relay peers. To fairly compare performance with flooding and random walk, we set their TTL such that the total search hop counts remain the same, and use this as an upper bound for BuSIS. In our emulation we set random walk TTL to be 25 given the average node connection degree is 5, then TTL for flooding is set to be 2. For multiple walkers in random walk, the total TTL can be split equally among the walkers. We also bound the maximum TTL for our incentive protocol to 25 for fairness.

Another important parameter is *participation payment*, whose setting is subtle and impacts the system functionality. Small participation payment leads to long warm up period of low system throughput due to shortage of credits for activating query searches. A new-join peer has no credit to issue query until earning enough participation payment. On the other hand, large participation payment will make peers ignore the earning from processing others' queries. It is because staying online earns enough credit to afford good search service. This undesirably stimulates the peers only to be

online but not contribute and serve others. Besides, large participation payment causes currency inflation as the total credits in the system grows unbound. We test the system throughput by varying the participation payment, the results is shown in Figure 3.2 based on which we set participation payment to be 5 for our emulation. Because this value is large enough to achieve good system throughput, but small enough to not generate query injection burst due to surplus credits.

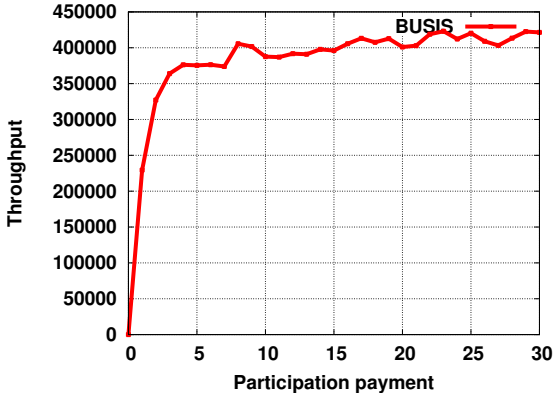

Figure 3.2: System throughput changes over participation payment

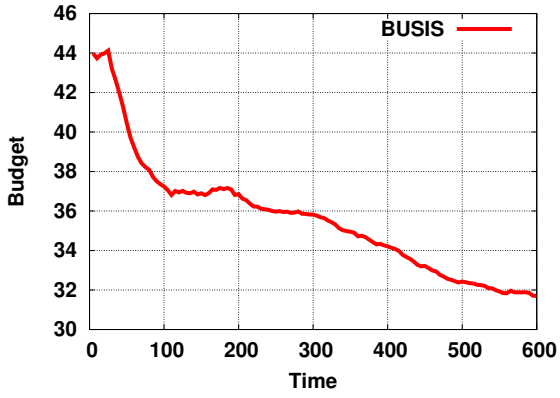

Figure 3.3: Budget adaptive trend over system operation time

3.3.3 Experimental Results

3.3.3.1 Performance Impact of Budget Assignment

In this emulation we examine the adaptivity and self-optimization capability of BuSIS in the face of changing object popularity. We emulate the object popularity changes by letting each peer perform searches and provide the search result as a replica to share with others. Initially each node holds 10% of total object, that is, an average object popularity is 0.1, and the object popularity increases as the system runs thus making the object search easier. The experimental results show that the average query budget decreases from 44 to 32 by 25% as shown in Figure 3.3, while the hit rate increases from 40% to 70% in Figure 3.4. The overhead per query is moderate from 6.5 hops to 5.5 hops as in Figure 3.5. The reason for such changes is that initially each peer does not have any knowledge on the input parameters, it takes the warm-up period to learn and update its estimation. Then assisted by the updated estimation, the utility function helps the peers to conduct the most beneficial search with high hit rate and low overhead. The self-optimizing and adaptability of BuSIS makes the search sensitive to the object popularity changes and always keep good performance with low overhead.

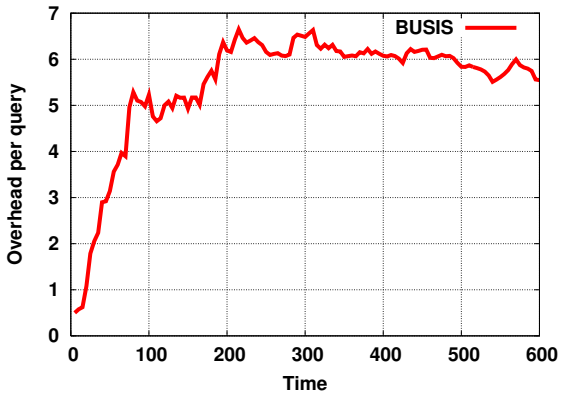

Figure 3.5: Overhead per query changing trend over system operation time

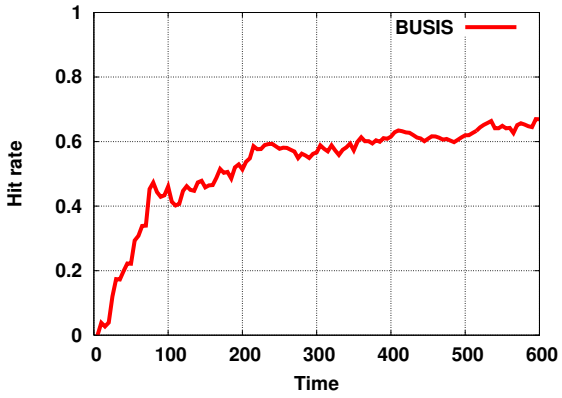

Figure 3.4: Hit rate changing trend over system operation time

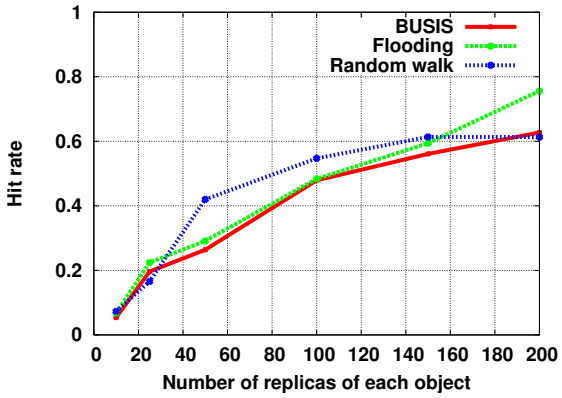

Figure 3.6: Hit rate under selfless user behavior

3.3.3.2 Performance Impact of User Behaviors

In this emulation we model selfless and selfish users by assigning different lifetimes, and compare the performance of BuSIS with that of Random walk and flooding. The selfless users keep on-line after they join the network, while the selfish users only join for searching file objects and leave once they get the objects. We first apply the selfless user behavior model, and compare the performance for the three search methods under fair TTL settings while adjusting the number of objects available per node. As shown in Figure 3.6, the hit rate of the three methods are similar, but the per query overhead sharply differs in Figure 3.7. Overhead is defined as the average number of hops used for each query. Flooding has the heaviest overhead, random walker has much less overhead than flooding and BuSIS has the lowest overhead, averaging around 25% of flooding and 50% of random walk. Especially when each user only holds a small percentage of the total objects (like 1%), BuSIS has remarkably reduced overhead while achieving the same hit rate. This scenario is common to the P2P file sharing systems, where each user only has a small number of files and needs to collaborate with others to get mutual benefits. The overhead reduction in BuSIS results from the self-optimization in the utility calculation, which optimizes the query budget to gracefully control the search scope according to the estimated object popularity.

When we apply the selfish user behavior model, BuSIS outperforms flooding and random walk by a larger amount than that in selfless user behavior model. As in Figure 3.8, the hit rate of BuSIS in most cases is higher than that of flooding and random walk. More importantly, comparing their results in Figure 3.9 with Figure 3.7 selfish behavior leads to dramatic overhead increase for both flooding and random

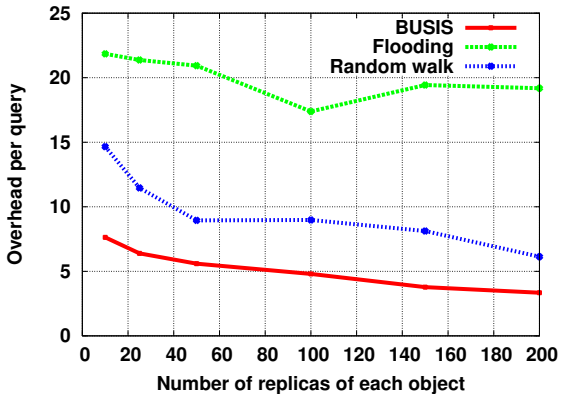

Figure 3.7: Overhead per query under selfless user behavior

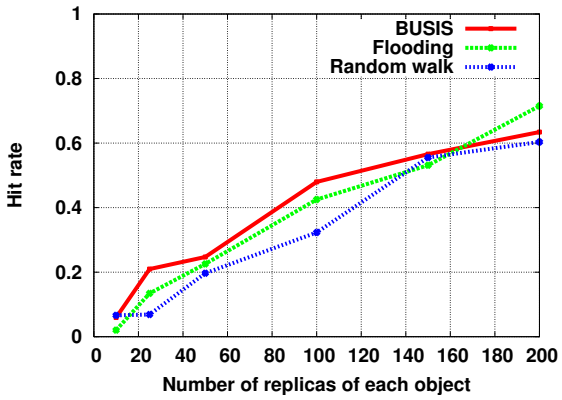

Figure 3.8: Hit rate under selfish user behavior

walk. In contrast, BuSIS keeps low overhead; especially when each node only has a small percentage of objects, our overhead is only 20% of flooding and 25% of random walk. This demonstrates that BuSIS always reduces the search overhead without hurting the hit rate, and is especially resilient to selfish user behavior. Flooding always has the largest overhead no matter whether users are selfless or selfish. Random walk has moderate overhead when users are selfless, but sharply increased overhead in case of selfish users, which is the common situation in real P2P file sharing systems.

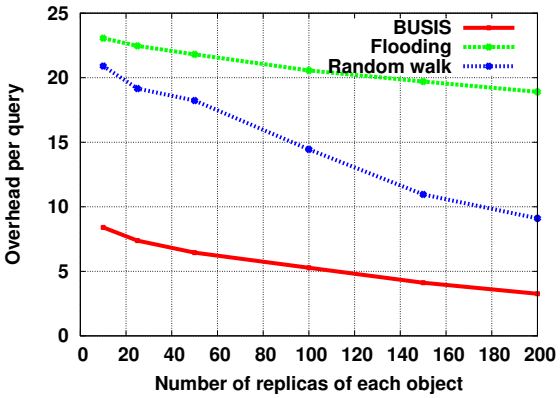

Figure 3.9: Overhead per query under selfish user behavior

3.3.3.3 Performance Impact of Dynamic Network Overload

In this emulation we examine the effectiveness of BuSIS in regulating user's behavior of query injection and moderating the query traffic load. We compare the query injection load by BuSIS and random walk under the same user demands, and the results are shown in Figure 3.10. To simulate user demands, after the warm up period at time $t = 300$ we put 100 queries at each peer's outgoing queue. The random walk produces the traffic burst around $t = 330$ due to the queuing delay and network conditions. In contrast, BuSIS does not generate traffic burst, the most intense traffic is around 1/3rd of random walk's. This smoothed traffic resulted from the query budget constraint in BuSIS. The peers are restricted from aggressively issuing queries, because they should wait until they earn enough credits to pay for the budget as decided by the utility function. So the traffic load is moderated over time by BuSIS.

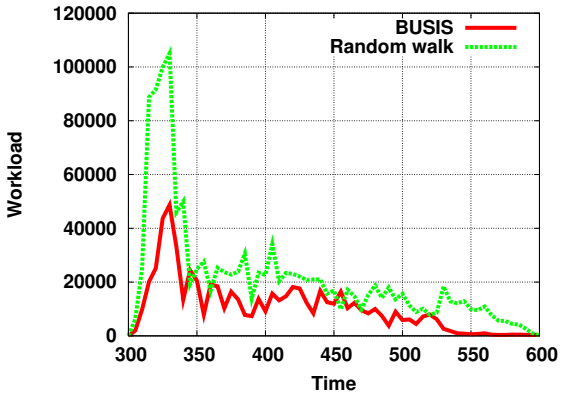

Figure 3.10: Workload changing over system operation time

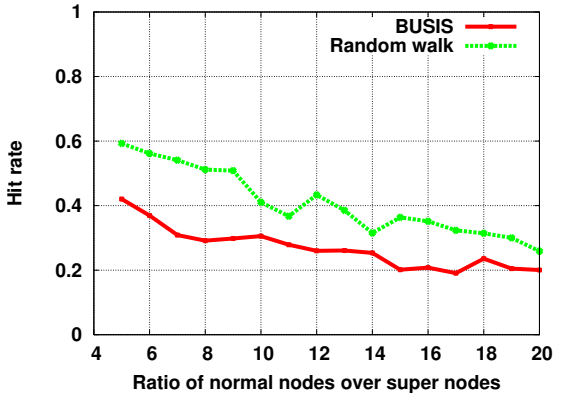

Figure 3.11: Hit rate of heterogeneous peers topology

3.3.3.4 Performance Impact of Heterogeneous Peers

All previous simulations only consider homogeneous peers, that is each peer on an average holds the same number of objects and has the same number of neighbor connections. Here we examine the heterogeneous peers (which is practical for the P2P systems), where each end user is of different bandwidth and processing capacity, and has different number of file objects. There are two kinds of nodes: normal nodes (termed as *n-nodes*) and super nodes (termed as *s-nodes*). Each normal node holds 10 objects and has around 4 to 5 neighbors, while a super node has 100 objects and 50 neighbors. We vary the ratio of normal nodes over super nodes and show the results of hit rate

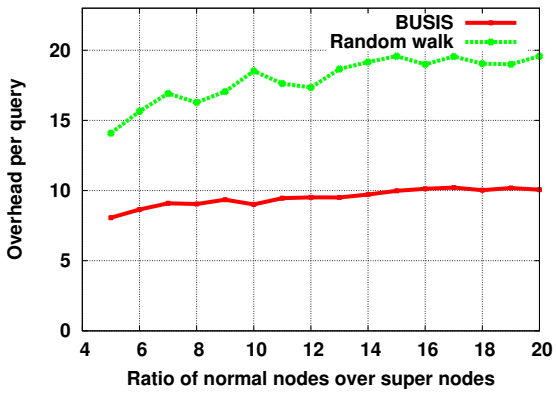

Figure 3.12: Overhead per query of heterogeneous peers topology

and overhead per query in Figure 3.11 and Figure 3.12 respectively. When the ratio of regular nodes over super nodes is small, random walk has higher hit rate than BuSIS but with much higher overhead, BuSIS reaches 2/3 hit rate of random walk with only 1/2 the overhead. This is because random walk search path easily converges at the super nodes, who aggregate the majority of the objects and make the common object search easier. However, this gain in performance requires a large portion of super nodes, around 20%, which is not the case in practice. When the portion of super nodes decreases, the hit rate of BuSIS catches up with that of random walk, while the overhead always stays at only half of the random walk. As shown in Figure 3.11 and Figure 3.12, when the percentage of super-nodes is around 5%, the hit rate of BuSIS is almost the same as random walk, but only with 50% overhead.

Credit distribution among heterogeneous peers is shown in Figure 3.13, where credit accumulations at a random s-node and at a random n-node are plotted. All nodes receive the same initial participation payment as 100 credit units for the first 100 seconds and generate the query at the same rate as 1 query per 5 seconds. The curve's fluctuations reflect earning credit by forwarding or replying queries and spending credits

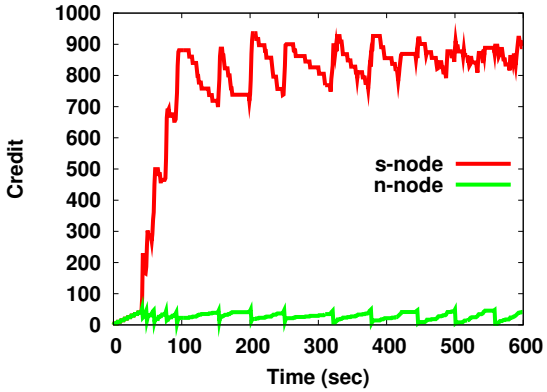

Figure 3.13: Credit distribution among heterogeneous peers

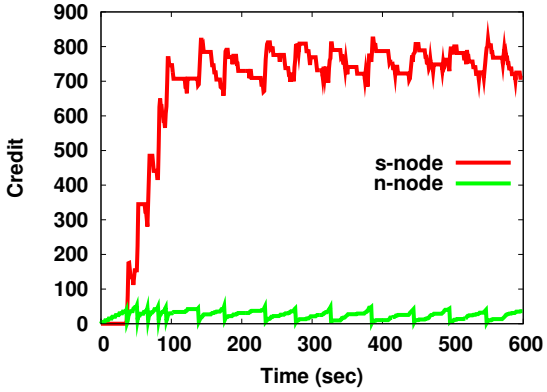

Figure 3.14: Differentiated participation payment for heterogeneous peers

in issuing qureies. After a brief warm-up period of 50 seconds, an s-node earns credit at a much faster pace than an n-node because of superior node connections and initial local files. As a result, an s-node issues queries more frequently, affords higher budget for each of its queries and accumulates more credits than an n-node. On the contrary, an n-node has to wait for earning enough credits to issue a query which has been generated and stored at its local buffer.

Results of differentiated participation payment for heterogeneous peers are shown in Figure 3.14, where each s-node does not receive any participation payment and each n-node receives 100 credit units for the first 100 seconds. The query genera-

tion rate is the same as before. The results of Figure 3.14 are similar to that of Figure 3.13 except that an s-node does not has any credit for the initial period. Once n-nodes start to issue queries by using their participation payment, s-nodes earn credit fastly and afford to issue query freqently. The results confirms that the pariticpation payment does not affect long-term or stable system performance but only serves to warm-up the system.

3.3.3.5 Discussions

BuSIS was designed to augment the random walk search with one-step implicit flooding by replicating neighbors sharing directories. This hybrid technique increases the hit rate of searching and also provides a greater chance for the peer's objects to be hit by others' queries to earn more credits. But this imposes more communication overhead for exchanging directories with neighbors. Especially, the overhead is heavier for updating the directories while the objects are replicated in more peers due to downloads. So we put this as an option for end users.

Chapter 4

Peer-to-Peer Indirect Reciprocity via Personal Currency

4.1 FairTrade Model

In this section, we first introduce the basic elements in FairTrade. Then, we present the Bayesian network model for peer credit setting and explain how a peer applies it.

4.1.1 Overview

In FairTrade, a P2P sharing system is modeled as a market. In this paper, we use a P2P file-sharing system as an example for illustrating FairTrade. A price is set for uploading a unit of data. The payment for downloading a file is the unit price times the file size. The requester peer pays to the uploader peer/peers for each file downloading.

The basic elements of FairTrade are as follows.

Personal currency. Instead of relying on a central bank to issue a global currency, each peer issues its personal currency in FairTrade which will be used as

payment. A peer's personal currency can be spent by the issuer or any other peer who received it before. The value of a personal currency is guaranteed by the issuer, as a peer must accept its personal currency as payment when offering services to others. The unit value of each personal currency is the same. But the acceptance of different personal currencies varies and depends on its issuer's capability and reliability to realize the value of its personal currency.

Peer credit limit. Each peer as a creditor sets the *peer credit limit* for accepting personal currencies issued by other peers as creditees. Creditor peer i sets a *peer credit* C_{ij} to a creditee peer j as the maximum amount of j's personal currency that will be accepted by peer i. The value of C_{ij} quantifies the trust peer i holds in peer j and depends on j's contribution and trading history with i or more reputable peers from i's perspective. *Peer credit* is asymmetric and is from the creditor's perspective, as C_{ij} is not necessarily equal to C_{ji}. C_{ij} will not be manually set by creditor peer i; FairTrade provides a Bayesian network model for dynamically setting C_{ij} on behalf of peer i.

Peer balance. At any given time, there is a peer balance between a pair of peers i and j who have been involved in a trading before. B_{ij} is the amount of j's personal currency that peer i is holding, and similarly B_{ji} is the amount of i's personal currency that peer j is holding. By the definition of *peer credit limit*, we have $0 \leq B_{ij} \leq C_{ij}$ and $0 \leq B_{ji} \leq C_{ji}$. The available amount of peer j's personal currency that can be accepted by peer i is $C_{ij} - B_{ij}$, while peer j can also use the amount B_{ji} of peer i's personal currency at any time.

Multilateral trading. In a trading, it is valid for the buyer peer (e.g., the downloader) to spend any personal currency it holds as payment as long as the issuer

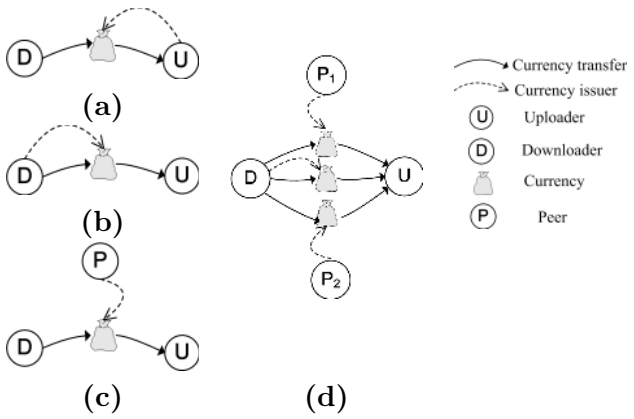

Figure 4.1: Types of Trading in FairTrade

has enough *peer credit* at the vendor peer (e.g., the uploader).

We define three basic types of trading in FairTrade: uploader guaranteed, downloader guaranteed, and third-party guaranteed. In an **uploader-guaranteed** trading, the downloader pays the uploader with the currency issued by the uploader, as shown in Figure 4.1(a). In a **downloader-guaranteed** trading, as shown in Figure 4.1(b), the downloader pays the uploader with the currency issued by itself. In a **third-party-guaranteed** trading, as shown in Figure 4.1(c), the downloader pays the uploader with the currency issued by a third peer. An actual trading may be any combination of the three basic types. Figure 4.1(d) shows an example in which the downloader pays the personal currencies issued by itself and two other peers.

The uploader accepts any amount of its personal currency when providing services. This is required for reciprocation, because other peers who hold the uploader's personal currency must have provided services to the uploader peer or on behalf of the uploader peer. With respect to other personal currencies, the uploader accepts up to an amount such that the issuer's peer balance does not exceed its peer credit limit.

4.1.2 Credit Setting in FairTrade

Each peer as a creditor sets the peer credit limit for every creditee peer periodically. All required information for setting the peer credit limit is collected and maintained by the creditor in a distributed way. There is no standard rule for setting the credit limit because it is a personal choice of the creditor. The setting calls for anticipating the probability that a creditee peer will cheat and evaluating the profit the creditee will bring [202]. On the conservative end, peer credit limit is set to a creditee peer to minimize the risk, while, on the liberal end, it is set to maximize the profit a creditee will produce. Peer credit limit can be any positive number, which induces infinite possibilities and complicates the problem.

To provide a practical solution in P2P systems, FairTrade simplifies the credit setting to a set of credit limit levels configured by the creditor peer. Each creditee peer is promoted one level at a time by accumulating successful trading experience with the creditor and other trustful peers from the creditor's perspective. Once the creditee cheats, its credit limit drops to zero. This level by level promotion gives the creditor adequate opportunities to learn the related characters of the creditee. The pace of promotion can be tuned to control the risk that the creditor is willing to take. Malicious peers are punished by losing all their peer credits, because the efficiency of the personal currency market is excessively degraded or even broken down due to participants' cheating [25].

In FairTrade, every peer gives a default amount of peer credit to all other peers. This default peer credit initializes the system without reliance on altruistic services. The amount of default peer credit is not sufficient to enable downloading an entire file, but helps a new peer start trading. Only after a new peer provides a service in return, does it

earn other personal currencies and build up its peer credit to complete the downloading.

4.1.3 Bayesian Model of Credit Setting

We transform the peer credit setting into a classification problem, which is solved by a Bayesian network model. Each peer as a creditor defines a set of credit limit levels in ascending order $S_L = \{L_0, L_1, \ldots, L_m\}$ where $L_i < L_{i+1}, 0 \leq i < m$. Each creditee peer starts from the default level L_0. L_m is the maximum peer credit a creditee may get from the creditor. The input for the credit setting is a set of character values of the creditee peer and its current peer credit limit level, and the output is divided into "Qualified" or "Disqualified" for promotion to the next level.

A set of variables $X = \{x_1, x_2, \ldots, x_k\}$ is defined as the influential characters of a creditee, where the values of these variables reflect the creditee's previous reciprocity with the creditor and other reliable peers from the creditor's perspective. Let $V_i (0 \leq i < m)$ denote the character value set of creditees at credit limit level L_i. V_i is partitioned into two subsets, V_i^Q and V_i^U. $x \in V_i^Q$ is the set of character values of a partner who qualifies for upgrading to level L_{i+1}, and $x \in V_i^U$ is the set of character values of a partner who does not qualify for a credit limit upgrade. It is impossible to classify everyone correctly, given that the information available for each peer is distributed collected from the dynamic P2P environment. FairTrade builds a Bayesian network model to minimize incorrect decisions and maintain a desirable level of consistency and continuity in the classification.

We use the following notation. For all variables, $0 \leq i < m$. x denotes a set of character values of a creditee.

- p_i^Q is the proportion of creditees with credit limit L_i who qualify for upgrading to L_{i+1}.

- p_i^U is the proportion of creditees with credit limit L_i who are not qualified for upgrading to L_{i+1}.
- $p_i(x|Q)$ is the probability that a creditee qualifies for L_{i+1} with character values x.
- $p_i(x|U)$ is the probability that a creditee is not qualified for L_{i+1} with character values x.
- $p_i(x)$ is the probability that a creditee with L_i has character values x.
- $q_i(Q|x)$ is the probability that a creditee of character values x qualifies for L_{i+1}.
- $q_i(U|x)$ is the probability that a creditee of character values x is not qualified for L_{i+1}.

We have conditional probabilities $q_i(Q|x) = \frac{p_i(x|Q)p_i^Q}{p_i(x)}$ and $q_i(U|x) = \frac{p_i(x|U)p_i^U}{p_i(x)}$.

Let C_i denote the profit loss from classifying a qualified customer as disqualified and rejecting its upgrade to L_{i+1}. The loss is proportional to the difference between the two levels of credit limit with coefficient c as $C_i = c \cdot (L_{i+1} - L_i)$. Let D_i denote the deficit incurred by classifying a disqualified customer as qualified and upgrading it to L_{i+1}. The deficit is proportional to L_{i+1} with coefficient d as $D_i = d \cdot L_{i+1}$. The expected total loss is given by Equation 4.1.

$$C_i \sum_{x \in V_i^U} q_i(Q|x)p_i(x) + D_i \sum_{x \in V_i^Q} q_i(U|x)p_i(x) \tag{4.1}$$

To maximize the set of qualified creditees with positive profits, the character values are classified as in Equation 4.2.

$$V_i^Q = \{x|D_i p_i(x|U)p_i^U \leq C_i p_i(x|Q)p_i^Q\} = \{x|\frac{p_i^U}{p_i^Q} \leq \frac{C_i p_i(x|Q)}{D_i p_i(x|U)}\} \tag{4.2}$$

We normalize the multivariate in x with common covariance; then Equation 4.2 is reduced to a linear rule as Equation 4.3, where y_i is the threshold character value

for creditees with L_i to upgrade to L_{i+1}.

$$V_i^Q = \{x | w_1 x_1 + w_2 x_2 + \ldots + w_k x_k > y_i\} \tag{4.3}$$

A linear scoring function is adopted to develop a classification rule due to its simplicity, robustness and efficiency compared with other forms (e.g., quadric), as proved in [145]. FairTrade defines a scoring function $s_i(x)$ for the set of character values x from creditees at each credit limit L_i as $s_i(x) = w_1 x_1 + w_2 x_2 + \ldots + w_k x_k$, where $w_1, w_2, \ldots, w_k$ are the impacting weights of the corresponding character variables. Now, the problem with k dimensions, represented by $p_i(x|Q)$, $p_i(x|U)$, is reduced to one dimension, represented by $p_i(s|Q)$, $p_i(s|U)$. Maximizing the set of qualified customers with positive profits expressed by Equation 4.1 is equivalent to finding the optimal cut-off y_i^* for the score as Equation 4.4.

$$y_i^* = \arg\min_{y_i} \{C_i \sum_{s<y_i} p_i(s|Q) p_i^Q + D_i \sum_{s \geq y_i} p_i(s|U) p_i^U\} \tag{4.4}$$

Figure 4.2 gives the overview of our Bayesian network model of credit setting.

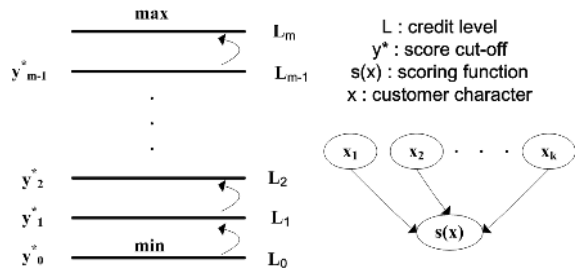

Figure 4.2: Credit Setting Model

4.1.4 Setting Credit with Bayesian Model

A cut-off score y_i^* is calculated periodically by a creditor for each level of peer credit limit $L_i (0 \leq i < m)$. To calculate the set of y_i^*, each creditor peer builds a

scoreboard accumulatively on the character values of all creditees. FairTrade instantiates three variables $X = \{x_1, x_2, x_3\}$ to essentially capture a creditee with low overhead.

- x_1 is the total amount of service the creditee has provided to the creditor. It is a direct indicator of a creditee's credibility.

- x_2 is the average value of the creditee's peer credit limit at other peers having higher peer credit limit at the creditor. Since a P2P system needs to make sure that the total consumption of peers is no more than their contribution to stabilize at a positive equilibrium, a creditor considers the creditee's contributions to others. While, only peers with higher peer credit limit are worth referring, so a malicious peer can only create fictitious peers with lower credit limit level. This is less beneficial than accumulating credit on its own, because every contribution is accounted only to one peer's credit history.

- x_3 is the credit limit level the creditor has at the creditee, which indicates the reciprocation degree between the creditee and the creditor.

The three variables are monotone increasing, so the higher the score of the scoring function $s_i(x) = w_1x_1 + w_2x_2 + w_3x_3$ the better. The impacting weights w_1, w_2, w_3 and the coefficients of loss and deficit c and d are estimated by the maximum likelihood method.

Each time a creditor completes a trading, it updates the x values of all involved creditees in the scoreboard. Every T time period, the creditor compares each creditee's score with the corresponding cut-off score to check whether a creditee qualifies for the next level of peer creditlimit .

4.2 FairTrade Design

In this section, we first describe the trading procedure in FairTrade. Then, we present the download schemes and uploader selection policies that have been applied in FairTrade.

4.2.1 Trading procedure

Figure 7.1 shows the procedure of a trade that consists of the following six steps.

1. A downloader peer i requests file f from a set of file holders H_f, which are discovered through a distributed hash table (DHT) lookup.

2. Each holder $j \in H_f$ replies to peer i with its price p_j, upload bandwidth b_j, estimated finish time t_j^f, red list Red_j, black list $Black_j$, and default credit limit L_{min}^j. The total download price for file f is the unit price p_j times the file size. Each peer j maintains an estimated finish time t_j for all tasks in its service queue. The estimated finish time for downloading file f from peer j is $t_j^f = t_j + size(f)/b_j$. The red list Red_j is a list of peers having high peer credit limit at j. The black list $Black_j$ is the list of peers having zero peer credit at j. The default credit limit L_{min}^j is the amount that j accepts for a new peer's personal currency.

3. The downloader peer i collects all replies from H_f and selects an uploader k by one of the rules described in Section 4.2.3.

4. The downloader peer i sends its payment – a set of personal currencies – to the selected uploader k for a validity check.

5. The selected uploader k checks with each issuer whose personal currency is paid

by i.

6. If the payment is valid, k acknowledges i with the payment acceptance. Otherwise, k notifies i of the invalid personal currency. i then returns to step 4.

Each node in FairTrade maintains a constant-size red list, which limits the communication overhead and sufficiently serves the purpose even in a large-scale P2P system. The reason is that a peer's interests are likely to cluster around a number of objects or topics as indicated by the power-law distribution. When a peer has established a stable trading experience with another peer, these two peers have high peer credit limit with each other, which makes them prefer continuing trade with each other. So each peer will have a relatively stable group of trading partner peers even when the system grows larger, and a constant-size red list captures these groups of stable trading partner peers. The size of the red list can be adjusted based on the network size.

A personal currency issued by a peer in FairTrade is a virtual currency, which is represented by the peer balance between the issuing peer and the accepting peer. Validating a peer currency means checking with both the issuing peer and the accepting peer for the peer balance between them and notifying them to update the peer balance if the peer currency is used.

4.2.2 Download Scheme

In FairTrade, the basic download scheme is to get a file from a single uploader, termed the *single-source* scheme. The *single-source* scheme is easy to implement but inefficient. In real networks, the bandwidth allocation is asymmetric because the uplink bandwidth takes only 1/10th of the downlink bandwidth in most cases. The downloader wastes its downlink bandwidth in the *single-source* scheme.

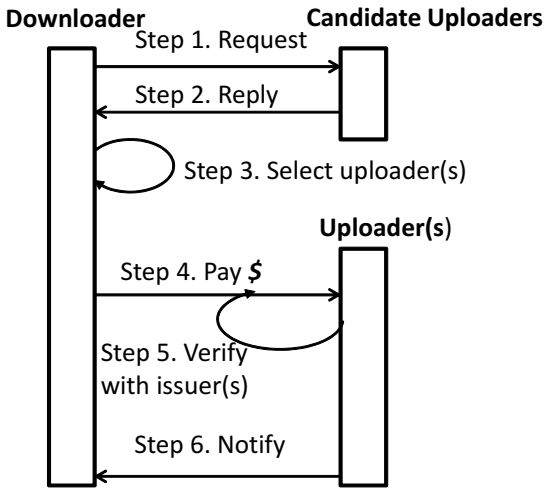

Figure 4.3: Procedures of a trading in FairTrade

We propose a *multiple-source* scheme to fully utilize the downlink bandwidth. In the *multiple-source* scheme, each file is divided into a number of segments and a downloader peer can download segments from multiple holders simultaneously. The *multiple-source* scheme also relaxes the peer credit requirement. When a downloader is short of personal currency to pay one uploader for an entire file, it may have enough personal currency for downloading some segments from one uploader and other segments from another uploader.

4.2.3 Uploader Selection Policy

When a downloader gets replies from a set of file holders, it needs to select one uploader or more. We present two rules for choosing uploader/uploaders in FairTrade.

Most-friendly selection. For a peer, its most-friendly uploader is the one from which it can download the most data using the personal currency in hand. Under the most-friendly rule, in the *single-source* scheme, the most-friendly peer in the set of file holders is selected as the uploader; and in the *multiple-source* scheme, the most-

friendly holder is selected as the major uploader and the remaining holders are used as assistant uploaders.

Fastest selection. The goal of the fastest selection is to minimize the download time of a task. In fastest selection, a downloader prefers an uploader from which it can finish the download soonest. In the *single-source* scheme, the fastest peer in the set of file holders is selected as the uploader. In the *multiple-source* scheme, the fastest holder is selected as the major uploader and the remaining holders are used as assistant uploaders. Among the assistant uploaders, data is preferentially downloaded from faster uploaders.

4.3 Attack Resistance Properties

In this section, we show that FairTrade is resistant to the three major types of attacks in P2P indirect reciprocity systems: sybil attacks, slander attacks, and whitewashing attacks.

Sybil-proof: A sybil attack [69] is when a peer creates a large set of identities (i.e., sybils), and directly modifies their peer credit limits to create an arbitrary contribution among them with the purpose of abstracting resources from the P2P network. We argue that this sybil strategy will not be profitable for the attacker. The reason is that the capacity of this network of sybils to extract resources from the remaining P2P network is bounded by the peer credit limits that these sybils have at normal peers, which is, in turn, bounded by the contributions that the sybils make to normal peers. There is no incentive to create large numbers of identities: no other peers will give peer credit for these sybils following the credit setting model in Section 4.1.4; thus, none of the sybils will get service from other normal peers.

Slander-proof: FairTrade is also designed to mitigate peer slander, where peers lie about the contribution they make to and receive from others. Since we require each pair of peers to agree on the peer credit limit and peer balance, no single peer can cheat on its contribution. Otherwise, it will not have any peer credit at other peers. Our credit setting model lets each peer build up its credit slowly through a history of honest trading.

To prevent double-spending and fake currency, FairTrade requires a validity check with the currency issuer before the currency is accepted. Whether the personal currency is spent by its issuer or other peers, its issuer should be online for the validity checking, otherwise it cannot be used. This in-person checking simplifies the procedure and imposes the online time requirement for peers who want to improve the acceptance of their personal currencies. Short-lived nodes, who cannot stay online long enough for other nodes to use their currencies, will not accumulate high peer credit at other nodes. Hence, the amounts of valid currencies issued by short-lived nodes are limited to the default amounts, which have negligible impact on the overall system performance. The majority of currencies exchanged in the system are issued by stable nodes based on our peer credit setting procedures. Thus, the in-person validation will not drag down the downloading process.

Whitewashing-proof: FairTrade is resistant to whitewashing attacks: the creation of fictitious identities, such that any penalties imposed by the system on a malicious peer is washed away. Since the only way to make whitewashing unprofitable is to make a newcomer and a heavily punished node indistinguishable, FairTrade prevents whitewashers by not offering any free service in the system. For each file, the default peer credit limit can only be used to download a fixed small segment of the file regardless of the identities of the downloaders or uploaders. Thus, a malicious peer with different

identities will only get the same small segment of a file repeatedly by using the default peer credit limit, which is useless on its own.

4.4 Performance Evaluation

4.4.1 Simulation Setting

We developed a P2P overlay simulator for FairTrade performance evaluation. A P2P file-sharing network is simulated with 10^4 peers. Each peer periodically generates requests to download movies, music files, games, and software in a catalog of $4 \cdot 10^4$ files. The file size is collected from real traces [6]. The cumulative distribution function (CDF) of file size is shown in Figure 4.4. We simulate the system for $7.2 * 10^5$ seconds in each simulation run and average over 20 runs to produce a result. The file popularity follows a Zipf distribution with parameter $\rho = 0.27$. The statistics of peer upload bandwidth is collected in the U.S. Broadband report [2], as shown in Figure 7.6, which ranges from 256 Kb/s to 10 Mb/s. In terms of willingness to share, we simulate two types of peers, as in [140]: 10% of the peers are content rich, each sharing 100 files; 90% of the peers are content lacking, each sharing 20 files. The initial sharing directory of each peer is a subset of files randomly selected based on the Zipf distribution. Each download file is cached in the requester's sharing directory. The file download request is generated at each peer following a Poisson process with average one request per 2000 seconds. Peers dynamically leave and join the system. When a peer reenters the system, it has the same sharing directory, peer credit limit, and peer balance as when it last left.

After generating a download request, the peer attempts to find an uploader via DHT routing. We use the Pastry overlay [170]. The peer may need to request the

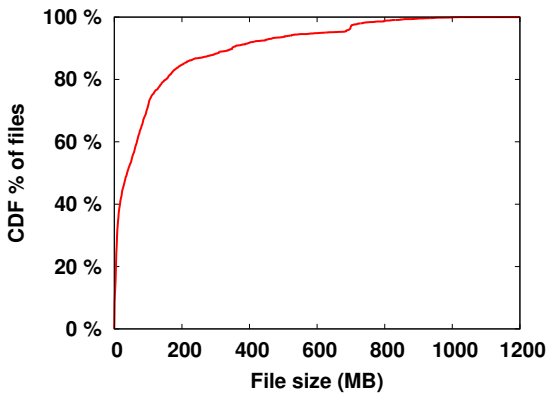

Figure 4.4: CDF of file sizes

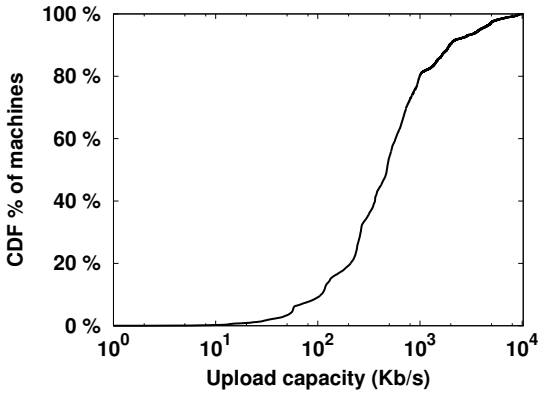

Figure 4.5: CDF of peers upload capacities

file several times, since it may not find a suitable uploader at the first attempt. Each peer buffers at most five unscheduled requests; a newly generated request is discarded when the buffer is full. As an uploader, each peer maintains a service queue to buffer all scheduled requests and serves them in order of their arrival times.

To simulate FairTrade peer credit setting, each peer sets a minimum credit limit level L_{min} (i.e., *default credit limit*), a maximum credit limit level L_{max} and a *stride* ξ, so the set of peer credit limit levels is $S_L = \{L_{min}, L_{min} + \xi, L_{min} + 2\xi, \ldots, L_{max}\}$. By default, L_{min} is set to 10, ξ is 5, and L_{max} is 200. Specific pricing schemes are out of the scope of the paper. We apply one uniform pricing and one non-uniform pricing as

example pricing schemes in FairTrade. For uniform pricing, a standard price p_0 is set for downloading a unit of data from any peer. For non-uniform pricing, each peer customizes its price based on p_0 and its upload bandwidth. Peer i's price $p_i = \frac{b_i}{b_{avg}} * p_0$, where b_i is its upload bandwidth and b_{avg} is the average upload bandwidth. The standard price p_0 in both cases eliminates exchange rate calculations between all personal currencies, since all of them are normalized with this common numeraire [25]. By default, the standard price is 1 unit of personal currency to download 1 MB data. The most-friendly uploader selection policy and the multiple-source download scheme are used by default.

4.4.2 Simulation Results

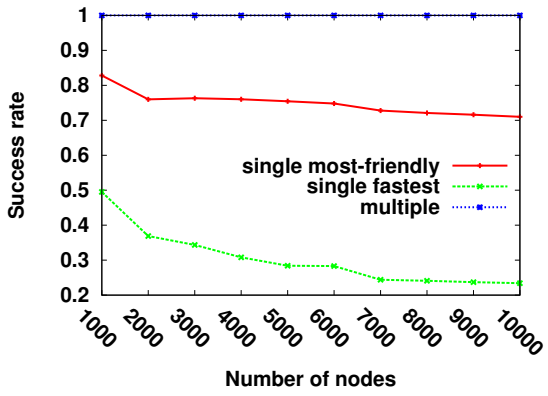

Figure 4.6: Success rates for non-uniform pricing

Success rate: We measure the success rates of different FairTrade policies while increasing the total number of nodes from 1000 to 10000. The *success rate* is the ratio of the number of successful requests over the sum of successful and unsuccessful requests. Figure 4.6 and Figure 4.7 show results of non-uniform pricing and uniform pricing. In the figures, the *single-source* and *multiple-source* schemes are abbreviated

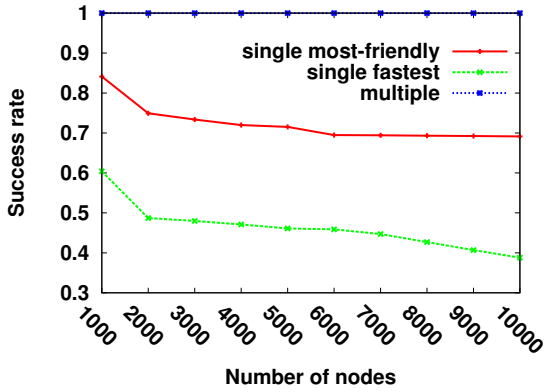

Figure 4.7: Success rates for uniform pricing

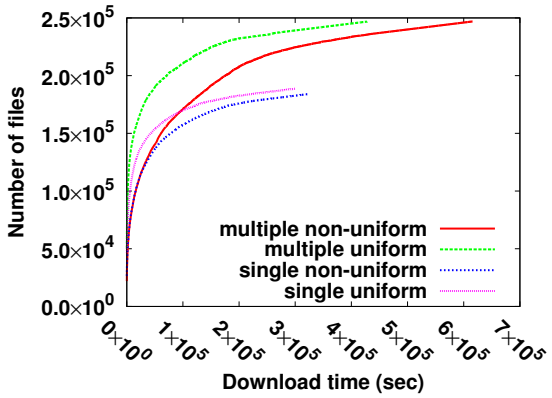

Figure 4.8: CDF of download times of various download and pricing schemes

as "single" and "multiple", respectively. *It is impressive that no matter which uploader selection policies or pricing schemes are used, the multiple-source scheme always keeps* 100% *success rate.* This is because the improved download bandwidth efficiency and relaxed credit requirement from the *multiple-source* scheme make downloading more flexible and faster, fewer requests are buffered than in the single-source scheme, and no request is discarded. The *most-friendly* uploader selection outperforms the *fastest* uploader selection in both pricing schemes, as can be seen by comparing the curves of "single most-friendly" and "single fastest" in Figure 4.6 and Figure 4.7. The reason is that credit requirement is the bottleneck for the *single-source* scheme. The *most-*

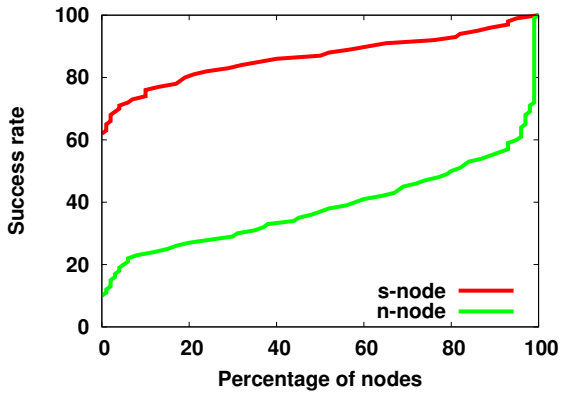

Figure 4.9: CDF of success rates of heterogeneous peers for *single-fastest* scheme

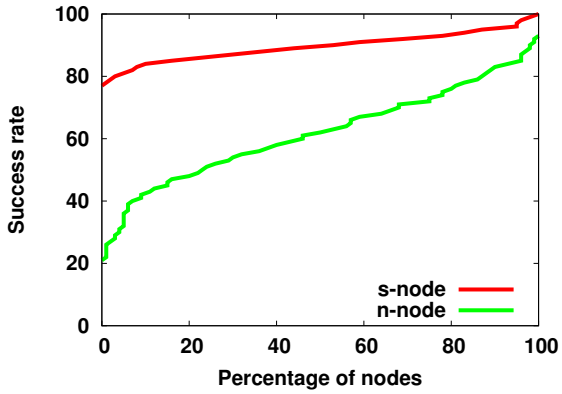

Figure 4.10: CDF of success rates of heterogeneous peers for *single-friendly* scheme

friendly selection targets downloading as much content as possible. When download success is not guaranteed, the download speed is not the most important factor, and *fastest uploader* selection does not benefit the success rate. The overall performance of uniform pricing is better than that of non-uniform pricing, especially for the *single-source* scheme with *fastest* uploader selection. The reason is that non-uniform pricing requires a longer wait to earn enough credit for downloading from a high-bandwidth peer. Longer waiting times cause more requests to be buffered and dropped. This is confirmed by the statistics of download times of non-uniform and uniform pricing, as shown in Figure 4.8.

To further understand the preferences of hetergeneous nodes on different downloading schemes, Figure 4.9 shows CDF of success rates of super nodes (termed as *s-nodes*) and normal nodes (termed as *n-nodes*) for *single-fastest* scheme, and Figure 4.10 shows that for *single-friendly* scheme. Since all nodes acheive 100% success rates at *multiple-source* schemes, the analysis on preferences only focuses on *single-source* schemes. Results show that s-nodes have much higher success rates in both schemes than n-nodes, and the differences between the two schemes are smaller for s-nodes than for n-nodes. For *single-fastest* scheme, the majority of s-nodes have 80% to 90% success rates, while the majority of n-nodes have 20% to 40% success rates. For *single-friendly* scheme, the majority of s-nodes have 85% to 95% success rates, while the majority of n-nodes have 40% to 70% success rates. s-nodes take advantage of larger inital local files and are more robust to different schemes while keeping high success rates. n-nodes have to rely on *multiple-source* schemes to achieve good success rates for breaking down the credit requirement is critical to them.

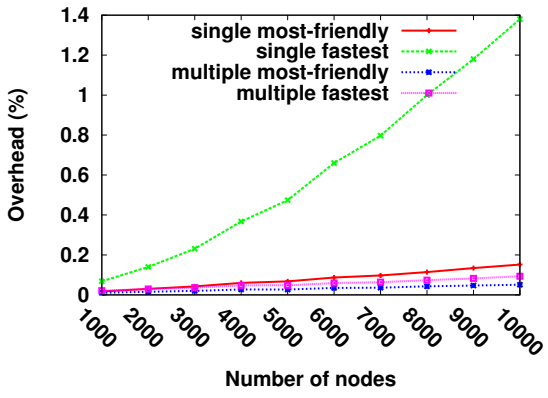

Figure 4.11: Trading overhead for non-uniform pricing

Trading Overhead: The *overhead* is measured as the ratio of maintenance traffic divided by download traffic. The maintenance traffic only counts the communi-

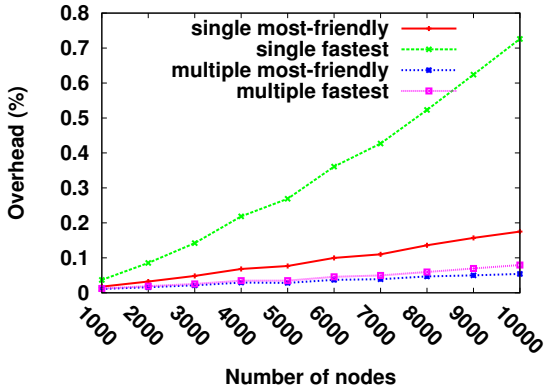

Figure 4.12: Trading overhead for uniform pricing

cations given in Figure 7.1. Figure 4.11 and Figure 4.12 show the results of non-uniform pricing and uniform pricing. The overheads of all schemes are under 1.5% because the maintenance packet size is much smaller than the content size. Only the overhead of the "single fastest" scheme noticeably increases as the total number of nodes increases, while the overheads of other three schemes remain almost the same. This is because the success rates of the three schemes remain the same as the total number of nodes increases. Thus, although the maintenance traffic increases, the download traffic also increases and therefore the ratio remains the same. But the success rate of the single fastest scheme drops as the total number of nodes increases, resulting in increased overhead.

Workload intensity: We vary the workload intensity by changing the request interval from 500 to 4000 seconds; a smaller interval gives a more intensive workload. The resultant success rates and overhead are shown in Figure 4.13 and Figure 4.14. The performance of all schemes is robust to workload intensity changes. This is because personal currency makes trading flexible: when more requests are generated, more op-

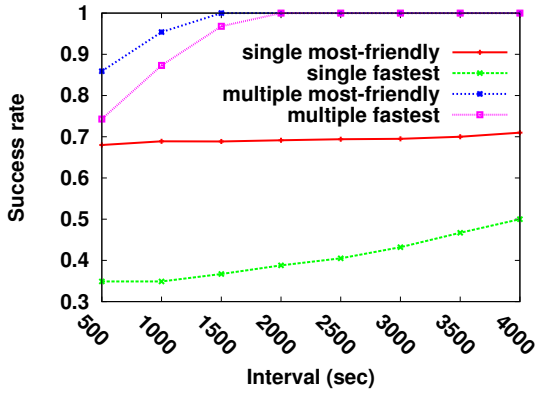

Figure 4.13: Success rates with various request intervals

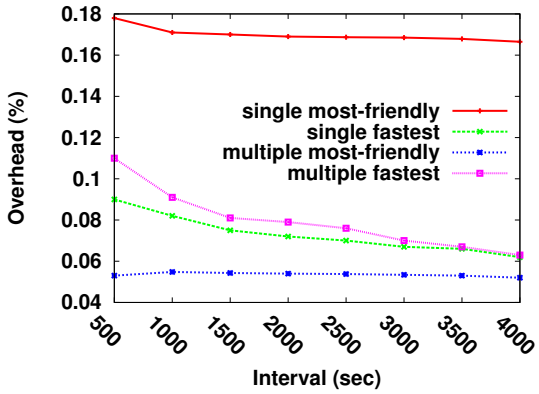

Figure 4.14: Trading overhead with various request intervals

portunities are opened for trading. The success rates of *multiple-source* schemes drop from 100% to around 85% when the workload is extremely intensive. This is because the total upload bandwidth of all peers is less than that required by the download requests in this case.

Warmup: The warmup period of FairTrade ends before peers issue their second requests, as shown in Figure 4.15, where each peer generates a request every 2000 seconds on average. The traffic stabilizes before 4000 seconds. We capture only the first 20000 seconds of the simulation to show the traffic trend. The small spike at the beginning stems from sending out buffered requests due to initialization. Issuing personal

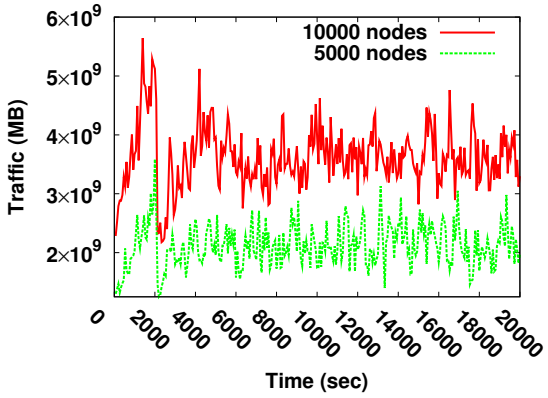

Figure 4.15: FairTrade warmup efficiency

currency enables newcomers to start trading more easily, and the system does not need to rely on altruistic services, especially for newcomers.

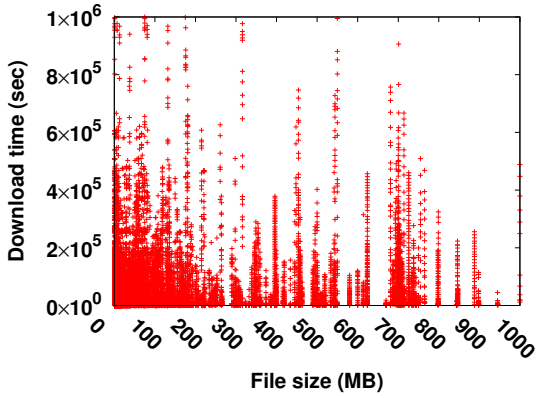

Figure 4.16: Distribution of download times using the *multiple-source* scheme

Download Time: The *download time* of a file is defined as the time between its request generation and download completion. As shown in Figure 4.12, the overhead is less than 1.5% of total traffic; the download time is also dominated by the file transfer time in FairTrade. Figure 4.16 and Figure 4.17 plot the distribution of download times using the *multiple-source* and *single-source* schemes, respectively. In these

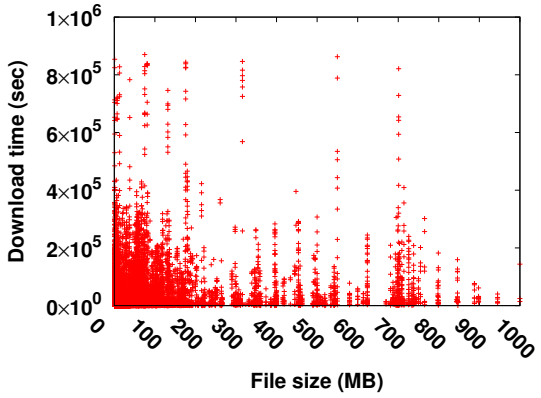

Figure 4.17: Distribution of download times using the *single-source* scheme

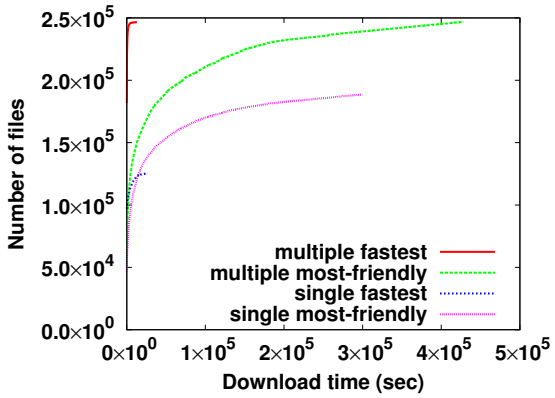

Figure 4.18: CDF of download times with various uploader selection policies

two experiments, we use uniform pricing and most-friendly uploader selection. The *multiple-source* scheme completes more downloads especially for large files which the *single-source* scheme cannot finish, due to extremely long download times. As shown in Figure 4.16 and Figure 4.17, the data points are more dense with the *multiple-source* scheme than with the *single-source* scheme.

Figure 4.18 shows the CDF of download times. We tried four different combinations of download schemes and uploader selection policies. The two selection policies have their own advantages. *Most-friendly* selection speeds up downloading when the *single-source* scheme is applied because the peer credit is usually a bottleneck in this

case. *Fastest* selection achieves the shortest download times when the *multiple-source* scheme is applied, where the peer credit is no longer a bottleneck.

Churn: In our simulation, each node follows an exponential distribution to independently decide when to leave. Peers enter and leave the system to make the total number of nodes stable. We vary churn rates, with probabilities of 10% to 50% of leaving during every 2000 seconds. By default, all results shown are simulated with 20% probability of churn. Performance with varying churn rates (not shown) is never less than the results shown by more than the width of the 95% confidence interval, so we do not show the results repeatedly. FairTrade is robust against churn because each trading in FairTrade does not rely on state information. New peers start up easily and rejoining peers resume quickly.

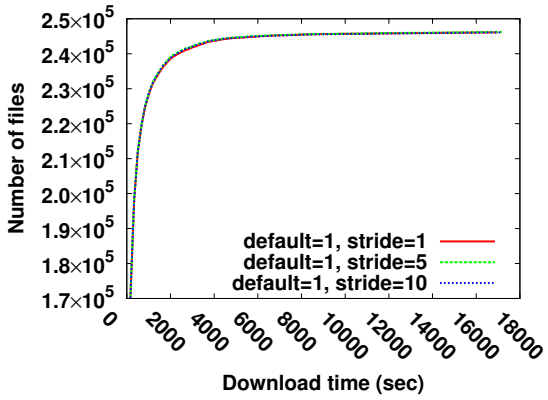

Figure 4.19: The impact of stride in credit setting without malicious nodes

Credit Setting: Credit setting is determined mostly by the default credit limit and the stride, which specify the amount of default credit limit level and the pace of credit limit increasing. Figure 4.19 and Figure 4.20 show their impacts on the download time. The success rates and overheads do not change significantly. The performance is

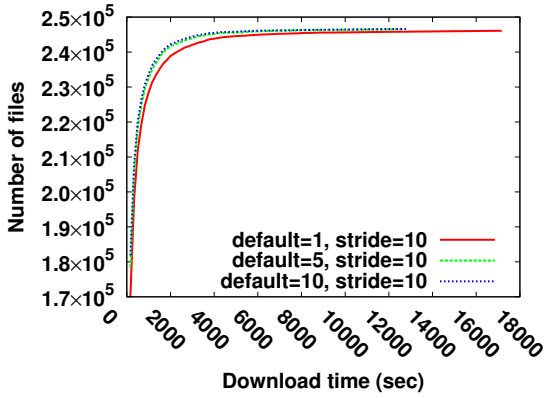

Figure 4.20: The impact of default credit without malicious nodes

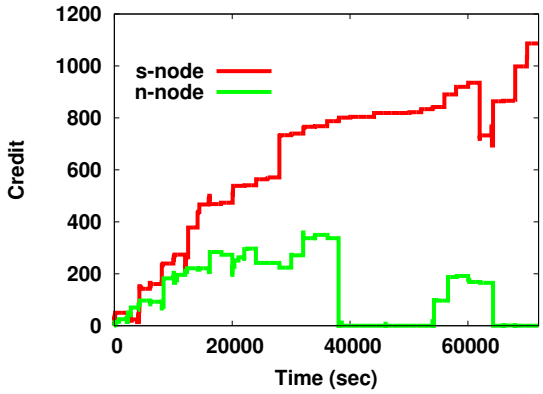

Figure 4.21: Credit distribution at heterogeneous nodes

slightly affected by the default credit limit but hardly affected by the stride. Increasing the stride from 1 to 10 does not change the results, as shown in Figure 4.19, while increasing the default limit from 1 to 5 reduces the download time, as shown in Figure 4.20. Further increasing the default limit to 10 yields much smaller improvement. This is because the default credit limit gives the startup fund for newcomers, which only affects the system warmup. After a peer starts trading, earning credit is not a problem for a non-malicious peer even when the peer credit limit increases slowly. Excessive default credit limit does not benefit the performance but fosters free-riding.

Credit Distribution: As described in experimental setting, two types of

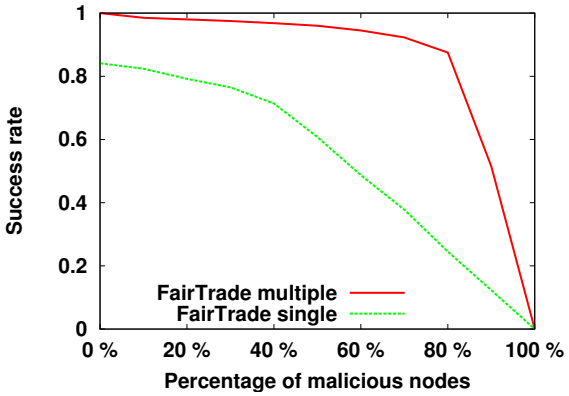

Figure 4.22: Success rate with malicious nodes

nodes are simulated: 10% of the nodes are content rich, each sharing 100 files (termed as *s-nodes*); 90% of the nodes are content lacking, each sharing 20 files (termed as *n-nodes*). Figure 4.21 plots credit accumulations at a random s-node and at a random n-node. The curve's fluctuations reflect earning credit by uploading files and spending credits in downloading files. After a brief warm-up period of 5000 seconds, an s-node earns credit at a much faster pace than an n-node because of richer initial local files. As a result, an s-node is able to download more files and accumulate more credits than an n-node. On the contrary, an n-node only can not accumulate spare credits but only make a balance between earning and spending credits.

Malicious Nodes: Malicious nodes are simulated as peers who reject their own personal currencies and do not provide service for others. We measure the success rate of non-malicious peers in the system as shown in Figure 4.22, where the population of malicious peers ranges from 0 to 100% of the total nodes. *Both FairTrade schemes are robust to malicious nodes because our credit setting not only considers the creditee's trading history with the creditor but also with other trustful peers.* As a result, the uploader as a creditor learns from others in addition to drawing lessons from past experience to quickly identify malicious nodes. The *multiple-source* scheme is more resilient

than the *single-source* scheme because the *multiple-source* scheme requires using other peers' personal currencies for downloading from assistant uploaders, which effectively prevents cheating.

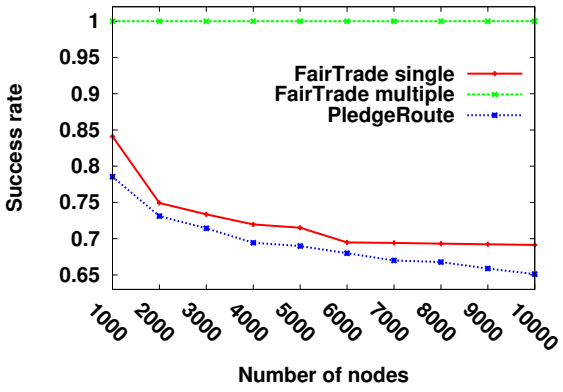

Figure 4.23: Success rate comparison

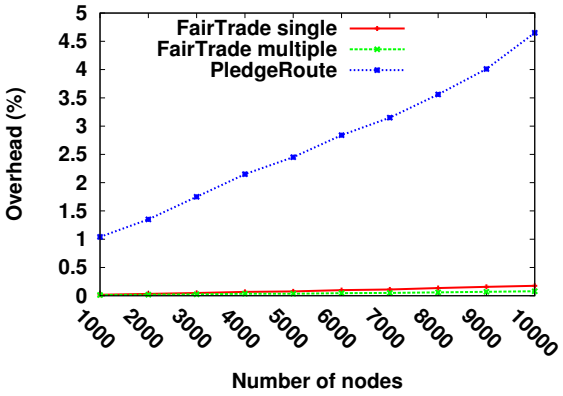

Figure 4.24: Overhead comparison

Comparison: We compare FairTrade with PledgeRoute [125], which is a recent P2P scheme designed for distributed indirect reciprocity. The core methods of FairTrade and PledgeRoute are described as follows.

- FairTrade: Each peer issues its personal currency that is used as payment for downloading in the system. A trade is successful when the downloader pays the

right amount of personal currencies whose issuers have enough peer credits at the uploader.

- PledgeRoute [125]: Each peer constructs its local view of the contribution network, which is a graphical representation of all contributions that have been given and received in the P2P system. A trading is successful when the downloader finds a path in its local view of the contribution network to the uploader with sufficient pairwise credit for contribution transfer.

Figures 4.23 and 4.24 shows the success rates and overhead of FairTrade and PledgeRoute with the total number of nodes ranging from 10^3 to 10^4. "FairTrade single" uses the *single-friendly* scheme and "FairTrade multiple" uses the *multiple-fastest* scheme. Both FairTrade schemes have higher success rates than PledgeRoute, because the personal currency enables directly paying other personal currencies for downloading. Compared to the hop-by-hop contribution transfer required by PledgeRoute, trading via personal currencies is more flexible and is more likely to success in FairTrade. The overhead of PledgeRoute increases more quickly than that of FairTrade because PledgeRoute requires more frequent topology discoveries of the contribution network as the system gets larger.

Chapter 5

P2P Indirect Reciprocity via Cooperative Banking

5.1 CoBank Scheme Overview

We give an overview of CoBank in this section. We begin with describing the basic procedure of a currency transaction in CoBank. Then, we list the issues that we target and explain how CoBank addresses these issues.

A P2P system is often modeled as a market for trading. Each peer earns currency by providing service and spends currency to get service. A currency transaction occurs when a peer – *buyer* buys service from another peer – *vendor*. In the transaction, the buyer pays an amount of currency to the vendor for the service.

Our scheme CoBank is designed to support the currency transactions between peers in a P2P system. We use a *global currency* to price the service system-wide. Each peer is associated with an account that records how much currency the peer possesses. Instead of having a central server that holds all peers' account data like in existing

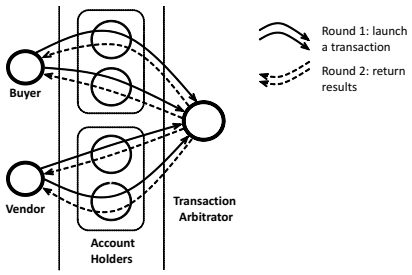

Figure 5.1: Basic procedure of a currency transaction in CoBank.

popular P2P systems [9, 4], we store and manage the account data in a distributed fashion, called *cooperative banking*. Each peer in the P2P system holds a small portion of account data, which is independent of the network size. For security purpose, the account data of each peer is decomposed into several parts, each of which is encoded and stored at a different peer. A peer is not responsible for managing the account data it holds. When two peers initiate a transaction, a third-party peer is selected as *transaction arbitrator*, which is responsible for managing and modifying the corresponding account data. CoBank uses distributed hash table (DHT) to locate the account data holders and transaction arbitrator for each transaction.

Figure 5.1 shows the basic procedure of a currency transaction in CoBank. When two peers initiate a transaction, they send requests to the peers that hold their account information. The account holders use the stored account information to validate these requests. If the requests are valid, the account holders then forward the account balance to the selected arbitrator. The transaction arbitrator aggregates the account information from different account holders and performs the transaction. Once getting the updated balances for both accounts, the transaction arbitrator decomposes the balances into parts and sends each part of the updated balances back to its holders. Finally, each account holder updates the part of account balance it stores and sends an

acknowledgment to the transaction initiator.

Next we briefly describe how the design of CoBank addresses the following issues - applicability (i.e., reducing the deployment requirements), scalability (i.e., supporting larger number of nodes in a P2P system), security (i.e., preventing various malicious attacks), and transaction atomicity (i.e., preventing a partially conducted transaction being exposed and protecting the integrity of a transaction).

Applicability. CoBank can be applied to all kinds of P2P applications since it uses *global currency*. *Global currency* does not have any special infrastructure requirement. For example, unlike pairwise currency [140, 162], *global currency* does not require a social network topology that gives the relationship information between peers. Besides, CoBank does not require any cryptographic infrastructure or central bank/broker, which is necessary for e-coin-based schemes [226, 223] to prevent the counterfeiters.

Scalability. CoBank is scalable with respect to the network size for the following three reasons. First, the communication overhead of each transaction grows logarithmically with the network size. In CoBank, the communication involved in each transaction is composed of a constant number of transmissions between overlay peers. If we use DHT as the overlay routing protocol, the communication overhead of each overlay transmission grows as the logarithms of the network size. Second, the storage overhead for maintaining account information at each node is a constant since the total size of account data is linear to the network size. Finally, as the management of the account data is taken care by the peers in a distributed fashion, there is no performance bottleneck in the P2P systems. For each transaction, the transaction arbitrator performs the transaction and the account holders actually store the updated account data. In CoBank, there is no central server for managing accounts or performing transactions.

Security. CoBank prevents the following attacks/abuses. First, CoBank can

prevent and detect slander attacks. With each peer's account data being divided and stored at different nodes, an account holder cannot misappropriate another peer's account since it does not hold the entire account information such as password. Since the arbitrators of a transaction only get the information related to the transaction and do not know the identity of the buyer or the vendor, they cannot misappropriate the accounts either. CoBank can also detect vicious falsification of account data by replicating the account data and using multiple arbitrators, which will be elaborated in Section 5.2.3. Second, CoBank can prevent sybil attacks [69] and white washings [125]. In CoBank, we do not give peers any startup fund. Each peer can earn currency by acting as an account holder and a transaction arbitrator. Therefore, creating new account does not give any benefit to the user. Our attack resistance properties are further analyzed in Section 5.4.

Transaction atomicity. CoBank ensures the *transaction atomicity* to protect fairness of each party in a transaction. We ensure the atomicity of each transaction by locking the related account data during the transaction. CoBank prevents a transaction being performed partially. A successful transaction must transfer the right amount of currency from the buyer's account to the vendor's account. No intermediate transaction state will be exposed in our scheme. With transaction atomicity, we also prevent peers abusing their account balance, e.g., double-spending [226].

5.2 Scheme Description

In this section, we describe our scheme – CoBank in details. First, we present the preliminaries and notations that will be used throughout the paper. Then, we formally define the procedure of a currency transaction in CoBank. Finally, two en-

hancement schemes are proposed, which further enhance the security of CoBank and provide incentives for peers to follow through our scheme CoBank.

5.2.1 Preliminaries and Notations

Table.5.1 summarizes the notations. In a currency transaction, we denote the buyer as B and the vendor as V. Each peer is associated with an account, which includes a peer ID, a password, and an account balance. For example, the buyer B's account consists of a peer ID ID_B, a password PSW_B, and an account balance ACC_B.

Each account is decomposed into k parts, where k is a parameter called *division factor*. Each part is stored at a peer, called holder. In other words, k holders cooperatively store a complete copy of an account. The holders are selected by hashing the peer ID of the account. For example, k holders for the account of peer B are determined by $hash(ID_B) = \{H_{1B}, H_{2B}, \ldots, H_{kB}\}$, where H_{iB} is the peer ID of the i^{th} holder. The password and balance of the account are encoded and divided into k parts, each of which is stored at an account holder for B. The original password is encoded and divided into a set of k parts as $f(PSW_B) = \{P_{1B}, P_{2B}, \ldots, P_{kB}\}$. Similarly, the account balance is encoded and divided as $f(ACC_B) = \{C_{1B}, C_{2B}, \ldots, C_{kB}\}$. Each holder stores the entire peer ID with a pair of password part and account balance part as $H_{1B} :< ID_B, P_{1B}, C_{1B} >, \ldots, H_{kB} :< ID_B, P_{kB}, C_{kB} >$.

Each transaction has a transaction ID Tr_{ID}. The transaction ID Tr_{ID} is generated by a random number generator taking inputs of the buyer ID, vendor ID and the time. The transaction arbitrator T is selected by hashing the transaction ID. For example, peer B (i.e., Buyer) wants to buy service from peer V (i.e., Vendor) at time t. The transaction ID Tr_{ID} is calculated by $random(ID_B, ID_V, t) =< Tr_{ID} >$ and the transaction arbitrator is selected by $hash(Tr_{ID}) = ID_T$, where ID_T is the peer ID of

the transaction arbitrator.

To enhance the account security, we replicate each account to have m complete copies stored at m sets of holders, where each set has k holders. The parameter m is called *replication factor*. We hash the account's peer ID to m sets of holder IDs, like peer B's m sets of holders are $hash'(ID_B) = \{H^1_{1B}, H^1_{2B}, \ldots, H^1_{kB}; \ldots; H^m_{1B}, H^m_{2B}, \ldots, H^m_{kB}\}$. The password and account balance are encoded and divided into k parts and paired into a set of k pairs, the same as for one set of holders. Each of the k pairs is replicated to m copies and stored at one holder of each set. For example, the first holder in each set stores the first pair and the m^{th} holder in each set stores the m^{th} pair, such as $H^1_{1B} :< ID_B, P_{1B}, C_{1B} >, \ldots, H^m_{1B} :< ID_B, P_{1B}, C_{1B} >; \ldots; H^1_{kB} :< ID_B, P_{kB}, C_{kB} >, \ldots, H^m_{kB} :< ID_B, P_{kB}, C_{kB} >$.

Similar to the account holder replications, we can select m arbitrators for each transaction by hashing the transaction ID to m peer IDs. Given a transaction ID Tr_{ID}, m transaction arbitrators can be selected by $hash'(TR_{ID}) = ID_{1T}, \ldots, ID_{mT} >$.

We define a reward interval with time duration Δ. Each user rewards its responsible account holders once every Δ time. The amount of bonus for each reward is $\$_{Bn}$. An irresponsible holder is fined $\$_{Fn}$ each time.

5.2.2 Basic Procedures

We use the example: at time t peer B wants to buy service from peer V at the price $\$_{Pay}$. For better clarity, we first describe the basic procedure by setting replication factor m to 1 and division factor k to 2. Thus, each account has one complete copy of the account information stored at a set of two holders. We explain the procedure with replicated account copies in Section 5.2.3.

Table 5.1: Summary of notations

Symbol	Definition
B	the buyer
V	the vendor
T	the transaction arbitrator
k	the number of parts that an account is divided into
m	the number of copies that an account is replicated
Tr_{ID}	the transaction ID
ID_B	ID of peer B
PSW_B	password of peer B
ACC_B	account balance of peer B
P_{iB}	the i^{th} part of B's password
C_{iB}	the i^{th} part of B's account balance
H^j_{iB}	the i^{th} holder in j^{th} set for B's account
$\$_{Pay}$	the amount of payment in a transaction
$\$_{Bn}$	the amount of bonus reward
$\$_{Fn}$	the amount of fine

STEP 1: The buyer and the vendor initiate a transaction by sending request to their account holders through DHT routing, respectively. The request includes the verification information (i.e., the peerID and the partial password), and the transaction information (i.e., the transaction ID and the payment amount).

- In step 1, buyer B sends the following information to its holders H_{1B} and H_{2B}: $B \overset{<ID_B,P_{1B},Tr_{ID},-\$_{Pay}>}{\longrightarrow} H_{1B}$ and $B \overset{<ID_B,P_{2B},Tr_{ID},-\$_{Pay}>}{\longrightarrow} H_{2B}$.

- Similarly, vendor V sends the following information to its holders H_{1V} and H_{2V}: $V \overset{<ID_V,P_{1V},Tr_{ID},+\$_{Pay}>}{\longrightarrow} H_{1V}$ and $V \overset{<ID_V,P_{2V},Tr_{ID},+\$_{Pay}>}{\longrightarrow} H_{2V}$.

STEP 2: After verifying the partial password, each account holder locks the account and forwards the transaction request to the arbitrator T through DHT routing. Each holder hashes the transaction ID Tr_{ID} to get the arbitrator ID T. The request includes the holder ID, the partial account balance, the transaction ID, and the payment amount.

- In step 2,H_{1B} and H_{2B} forward the transaction request to the arbitrator T:

$H_{1B} \xrightarrow{<ID_{H_{1B}}, C_{1B}, Tr_{ID}, -\$_{Pay}>} T$ and $H_{2B} \xrightarrow{<ID_{H_{2B}}, C_{2B}, Tr_{ID}, -\$_{Pay}>} T$.

- Similarly, H_{1V} and H_{2V} forward the transaction request to the arbitrator T: $H_{1V} \xrightarrow{<ID_{H_{1V}}, C_{1V}, Tr_{ID}, +\$_{Pay}>} T$ and $H_{2V} \xrightarrow{<ID_{H_{2V}}, C_{2V}, Tr_{ID}, +\$_{Pay}>} T$.

STEP 3: The transaction arbitrator combines all parts of account balance to get the original account balance for both the buyer and the vendor. It then checks whether the buyer has enough currency to pay. If the buyer has sufficient currency, the arbitrator transfers the payment from the buyer's account to the vendor's account and sends back the updated account balances to each account holder. Otherwise, the arbitrator aborts the transaction and notifies each account holder of the transaction failure. The arbitrator directly contacts back with each holder and does not need DHT routing.

- T combines C_{1B} with C_{2B}, decodes the result to get the account balance ACC_B, and checks whether the buyer has enough currency to pay (i.e., whether ACC_B is larger than $\$_{Pay}$).
  - If $ACC_B > \$_{Pay}$, T also combines C_{1V} with C_{2V} to get the account balance ACC_V. T then transfers $\$_{Pay}$ from ACC_B to ACC_V, i.e., $ACC_B = ACC_B - \$_{Pay}$, $ACC_V = ACC_V + \$_{Pay}$.
  - Otherwise, T terminates the transaction and notifies the account holders of both parties, i.e., $T \xrightarrow{<Tr_{ID}, failure>} \{H_{1B}, H_{2B}, H_{1V}, H_{2V}\}$.
- T encodes the updated accounts of both parties, divides them, and sends each part to the corresponding account holder. $T \xrightarrow{<C_{1B}, Tr_{ID}, OK>} H_{1B}$, $T \xrightarrow{<C_{2B}, Tr_{ID}, OK>} H_{2B}$, $T \xrightarrow{<C_{1V}, Tr_{ID}, OK>} H_{1V}$, and $T \xrightarrow{<C_{2V}, Tr_{ID}, OK>} H_{2V}$.

STEP 4: The account holders directly send back the transaction result to the buyer and the vendor, respectively. Each account holder then unlocks the account.

- If the transaction succeeded, the updated account information is sent back with transaction ID. $H_{1B} \xrightarrow{<C_{1B}, Tr_{ID}, OK>} B$, $H_{2B} \xrightarrow{<C_{2B}, Tr_{ID}, OK>} B$, $H_{1V} \xrightarrow{<C_{1V}, Tr_{ID}, OK>} V$, and $H_{2V} \xrightarrow{<C_{2V}, Tr_{ID}, OK>} V$.

- If the transaction failed, the original account information is sent back with the lack of fund notification. $H_{1B} \xrightarrow{<C_{1B}, Tr_{ID}, failure>} B$, $H_{2B} \xrightarrow{<C_{2B}, Tr_{ID}, failure>} B$, $H_{1V} \xrightarrow{<C_{1V}, Tr_{ID}, failure>} V$, and $H_{2V} \xrightarrow{<C_{2V}, Tr_{ID}, failure>} V$.

5.2.3 Enhancement with Replication

To enhance the account security, we replicate the account information of each peer to have m (i.e. the replication factor) copies. Each account then has m sets of account holders. We also use m arbitrators for each transaction. Figure 5.2 shows a currency transaction procedure in CoBank with replications, where the replication factor $m = 3$. Each cloud represents a single set of account holders. The transaction procedure has the same four steps as the basic one as shown in Section 5.2.2, except that the buyer and vendor send transaction requests to all m sets of account holders. Each account holder forwards request to all m arbitrators. Then, each arbitrator uses the majority rule to detect any irresponsible holders and obtain the correct account balance of both parties. After a transaction is performed by an arbitrator, the transaction result is sent back to each holder in all m sets. In this way, each account holder receives results from m arbitrators so that any misbehavior of an arbitrator can be detected and corrected. To carry out the majority rule, m is at least 3.

Each transaction arbitrator uses a timeout T_{out} to avoid endless waiting for

missing requests from some account holders. Such missing requests may be caused by packet loss or node failures. After receiving the first arrived transaction request from an account holder, an arbitrator starts the timeout T_{out}, and stops waiting for other not-yet-arrived requests when T_{out} expires. As long as an arbitrator receives requests from a predefined minimal number of account holder sets, it performs the transaction; otherwise, it aborts the transaction. If an application has special security concerns, the minimal number of sets required to perform a transaction may be set larger as well as the replication factor. The timeout T_{out} is set to cover a round-trip-time (RTT) between an account holder and an arbitrator on average, which is equal to the average hops in a DHT routing multiplied by the RTT for each hop.

With replication, the communication overhead for each transaction is $O(k * m^2 \log N)$, where N is the total number of peers, k is the division factor and m is the replication factor. Since the values of k and m are small constants (e.g., $k = 2,\ m = 3$), CoBank keeps $O(\log N)$ communication overhead for each transaction. The storage overhead for each account is no more than $m * k$ times an account size, which is still a small constant.

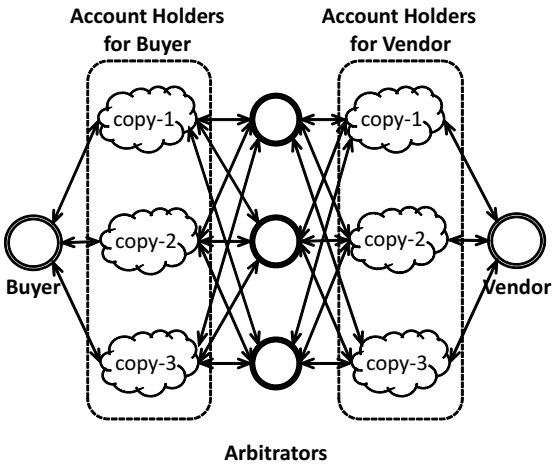

Figure 5.2: A currency transaction procedure with replications

5.2.4 Incentives for Participating in CoBank

Peers may lack interests to take part in the distributed account and transaction management in our scheme. We offer monetary incentives by rewarding responsible holders periodically, and punish any irresponsible holder or arbitrator every time it is detected.

5.2.4.1 Reward

Periodically each peer rewards its account holders, who are reliably taking the role during the period, with an amount of bonus $\$_{Bn}$. The length of the reward period is large enough to balance off the communication overhead for the reward transaction and justify the amount of bonus. The basic procedure for rewarding is as follows: when peer B wants to reward its account holder H_{1B}, B sends a bonus certificate to all H_{1B}'s account holders $H_{1H_{1B}}, H_{2H_{1B}}$ as $B \overset{<ID_B, ID_{H_{1B}}, Tr_{ID1}, \$_{Bn}>}{\longrightarrow} H_{1H_{1B}}, H_{2H_{1B}}$. The reward transaction is performed by the arbitrator T_1, whose ID is hashed by the transaction ID Tr_{ID1}. The arbitrator T_1 increases the account balance of H_{1B} by $\$_{Bn}$, and then

sends back the updated parts to $H_{1H_{1B}}, H_{2H_{1B}}$, respectively. Account holders $H_{1H_{1B}}$ and $H_{2H_{1B}}$ then notifies H_{1B} and B of the reward result.

5.2.4.2 Punishment

In each transaction, each of the m arbitrators applies the majority rule, and detects irresponsible account holder if there exists any. An account holder is marked irresponsible because of cheating on the account information or not participating. The punishment message is encoded by the arbitrator who detected this and sent back to the account owner (i.e., either the buyer or the vendor in the transaction) through piggyback on another good account holder. After receiving the punishment message, the account owner initiates a punishment transaction in the same way as the reward transaction, except deducting an amount of $\$_{Fn}$ from the holder's account rather than adding an amount of $\$_{Bn}$.

For each transaction, an account holder also applies the majority rule on the results received from m arbitrators so that any misbehaved arbitrator is detected and punished by a fine of $\$_{Fn}$. Each account holder, who detected an arbitrator misbehavior, reports to the account owner (i.e., the buyer or the vendor of the transaction) and synchronizes the result with other account holders. The account owner initiates a punishment transaction for a bad arbitrator in the same way as for a bad account holder.

5.3 Analysis of Node Selections

A sufficient level of CoBank efficiency is ensured by reliable account holders and transaction arbitrators (termed as *accounting nodes*), since every trade requires par-

ticipations of related accounting nodes. Thus, we select reliable nodes to be accounting nodes. In this section, we analyze node reliability, examine resource consumptions for being an accounting node, and optimize the selection threshold constraint by resource budget. We analyze the most scarce resource – the bandwidth consumption in our model.

5.3.1 Node Availability Model

We propose a distributed scheme to select the most stable peers with adequate bandwidth to be accounting nodes, which form a new DHT called *accounting DHT (A-DHT)*. All the nodes in CoBank are organized into a DHT by default, and use their ID to hash into accounting DHT for locating their account holders and transaction arbitrators.

To estimate the number of qualified accounting nodes, we analyze the distribution of peers' availability.

We consider a system with n nodes in steady-state. Each node switches between two states: ON when the node gets online during its uptime and OFF when the node goes offline during its downtime. A node's uptime is the time interval between joining and leaving. Its downtime is the time interval from departure to re-joining. $U(\cdot)$ is the cumulative distribution function (CDF) of the node uptime, and $\bar{u}$ is the average uptime. $D(\cdot)$ is the CDF of the node downtime, and $\bar{d}$ is the average downtime. A node's availability a is measured by the probability that a node is in the ON state, $a = \frac{\bar{u}}{\bar{u}+\bar{d}}$. The expected number of nodes in the ON state is an.

We examine nodes with expected uptime above T. The information about the node uptime can be obtained either by sampling or by prediction techniques [149, 148]. Let n' be the number of nodes above the threshold T and in their uptime, $n' \leq n$. $\rho = \frac{n'}{an}$ represents the ratio of such nodes, $0 < \rho \leq 1$. $U'(\cdot), D'(\cdot), \bar{u}', \bar{d}'$ and a' represent

the counterpart attributes of the nodes with expected uptime above T as $U(\cdot), D(\cdot), \bar{u}, \bar{d}$ and a of all nodes, respectively. Our analysis assumes a node maintains $\log(n')$ overlay connections which is the same as with the general DHT.

Measurement results [175, 130] demonstrate that node uptime is well modeled by a long-tailed distribution. We adopt a shifted Pareto distribution to depict the independence of node uptime as [214]. The probability density function (PDF) of node uptime $u(t)$ is shown in Equation 7.1, where $\alpha > 1, \beta > 0$.

$$u(t) = \frac{\alpha}{\beta}(1 + \frac{t}{\beta})^{-(\alpha+1)} \tag{5.1}$$

And the CDF of the node uptime $U(t)$ is shown in Equation 7.2, where $\alpha > 1, \beta > 0$.

$$U(t) = 1 - (1 + \frac{t}{\beta})^{-\alpha} \tag{5.2}$$

The smaller value of α means a stabler system with longer node uptime. The PDF of the selected A-DHT node uptime $u'(t)$ is derived in Equation 7.3.

$$u'(t) = \begin{cases} 0 & t < T \\ \frac{u(t)}{\int_{t=T}^{\infty} u(t)dt} = (1 + \frac{T}{\beta})^{\alpha} u(t) & t \geq T \end{cases} \tag{5.3}$$

And the CDF of the selected A-DHT node uptime $U'(t)$ is derived in Equation 7.4.

$$U'(t) = \begin{cases} 0 & t < T \\ \frac{U(t)-U(T)}{\int_{t=T}^{\infty} u(t)dt} = (1 + \frac{T}{\beta})^{\alpha}(U(t) - U(T)) & t \geq T \end{cases} \tag{5.4}$$

The availability of the selected A-DHT nodes is measured by their average

uptime $\bar{u'}$ derived in Equation 7.5.

$$\bar{u'} = \int_{t=T}^{\infty} t \cdot u'(t)dt = \frac{\beta + \alpha T}{\alpha - 1} \tag{5.5}$$

The higher the threshold T, the better the system stability $\bar{u'}$. This system stability determines the frequency of maintenance probing and thus the expected overhead.

To calculate the ratio of selected A-DHT nodes over all nodes, we compute the values of n' and ρ as a function of the selection threshold T. According to Little's Law, the average number of nodes in a stable system equals their average arrival rate multiplied by their average uptime in the system. Applying to all nodes in the ON state, we get $an = \lambda\bar{u}$, where λ is the average arrival rates of nodes to the ON state. Applying Little's Law to the selected A-DHT nodes in the ON State, we get $n' = \lambda(1 - U(T))\bar{u'}$. Substituting the variables $an, n', U(T)$ and using $\bar{u} = \int_{t=0}^{\infty} t \cdot u(t)dt = \frac{\beta}{\alpha-1}$, the percentage of selected A-DHT nodes ρ is derived in Equation 7.6, which estimates how many nodes are qualified for the threshold T.

$$\rho = \frac{n'}{an} = (1 + \frac{T}{\beta})^{-\alpha}(1 + \frac{\alpha}{\beta}T) \tag{5.6}$$

5.3.2 Resource Consumption Analysis

We examine the bandwidth resource consumption for being an accounting node, which comes from three aspects: workload for participation in trades as an account holder B_1, workload for participation in trades as a transaction arbitrator B_2, maintenance workload in A-DHT B_3. We use q_r to denote the average trade rate initialized by a node, d_c to denote the average size of an account information stored at an account holder, d_m to denote the average size of maintenance package in A-DHT, r_c to denote the number of replicas for an account information stored in A-DHT. We sets the

probe frequency for checking the availability of A-DHT neighbors to be $\frac{A}{u'}$, A is a small constant $A > 1$.

$$B_1 = \frac{an}{n'} kmq_r 4d_c \tag{5.7}$$

$$B_2 = \frac{an}{n'} mq_r 2kd_c \tag{5.8}$$

$$B_3 = \frac{A}{u'} \log(n') d_m + \frac{an}{n'} kmd_c r_c \frac{1}{u'} \tag{5.9}$$

The total bandwidth consumption for being an accounting node is $B_1+B_2+B_3$.

5.3.3 Availability Threshold Setting

We set the availability threshold T^* as in Equation 7.10 to select the most reliable nodes as accounting nodes constrained by their available bandwidth resource constraint. We use b_w to denote the bandwidth budget per node used for being accounting nodes.

$$T^* = arg \ \max T \ \ s.t. \ B_1 + B_2 + B_3 \leq b_w \tag{5.10}$$

5.4 Attack Resistance Properties

In this section, we show that CoBank is resistant to all three major types of attacks jeopardizing P2P indirect reciprocity systems – sybil attacks, slander attacks and whitewashing attacks. We address each type of attacks by considering all possible attackers in the context of CoBank: buyers, vendors, account holders, transaction arbitrators, and all other peers.

5.4.1 Resistance to Sybil Attack

The sybil attack [69] takes the following form: a user creates a large set of identities (i.e., fictitious users) and makes up their account balances to create arbitrary contributions among them. The attacker then tries to use the invented contributions to extract extra resources from the network.

CoBank prevents the sybil attack for the following two reasons. First, the account information of a peer is stored at a set of account holders. All the account holders are selected through hashing the peer ID. Thus, the user cannot forge its account balance. Second, CoBank does not need to provide startup fund for newcomers since it provides incentives for participating in CoBank. Thus, users cannot gain any benefit by creating an additional user identity. Therefore, we prevent sybil attacks by making a user cannot take advantage of the attacks.

5.4.2 Resistance to Peer Slander

The slander attack consists of all cases that a user lies regarding its contribution or contributions of others. We list all possible cases in CoBank and discuss how we mitigate them one by one.

- **A user cheats on its account balance**. CoBank prevents this case by initializing each new comer's account balance to 0 and distributing the information to a set of account holders. In the user's transactions, its account information is directly retrieved from and updated to the account holders. Thus, the user cannot modify its account to cheat. Moreover, CoBank also prevents a user "double spending" the currency, since we use virtual currency (i.e. the account balance) rather than the e-coins or e-currency as in [226, 223].

- **An account holder cheats on the account balance stored**. First, CoBank prevents an account holder impersonating an account owner, since each account holder only stores partial information of the account password and balance. No single account holder has the complete account information. Second, CoBank prevents an account holder lying about the account balance by replicating multiple sets of account holders. In each transaction, an arbitrator will compare the account balance sent from each set of holders and detect the cheaters.

- **A transaction arbitrator abuses the account balance for either of the two transaction parties**. CoBank prevents this case by hiding the account owner ID and password from the arbitrator, who has no way to impersonate the account owner or reuse the account balance. Besides, for each transaction each account holder receives results from m arbitrators, thus any misbehaviored arbitrator can be detected by the majority rule.

5.4.3 Resistance to Whitewashing

The whitewashing attack is that a user creates disposable identities and discards any identity that has been labeled as malicious by the system. In this way, the penalty imposed on malicious user is whitewashed.

CoBank prevents the whitewashing attacks for the following two reasons. First, we penalizes the malicious account holders and arbitrators by deducting a fine of $\$_{Fn}$ from their account balance every time. When a peer's account balance is penalized to zero or below, we block the peer either through the DHT application-specific security policies [238] or marking its original information that is used to assign its peerID when the peer first joining the DHT, such as IP address [170]. We also store the blocking

information in the DHT, which only adds a negligible storage overhead. Second, we do not provide any extra fund to new comers. There is no advantage for whitewahsers to abandon a currently used identity with non-zero account balance and rejoin again. Therefore, our scheme does not suffer from whitewashing attacks.

5.5 Performance Evaluation

In this section, we evaluate our scheme CoBank using a file-sharing application. While file-sharing is useful as a tangible example, the operations in CoBank can be used equally well for other kinds of resources, such as storage spaces in a backup system, messages in a publish subscribe system, or the results of a computation in a grid computing system.

5.5.1 Simulation Setup

5.5.1.1 Simulation Methodology

We developed a P2P overlay simulator for CoBank performance evaluation. A P2P file sharing network is simulated with 10^4 peers. Each peer periodically generates requests to download movies, music files, games and softwares in a catalog of $4 \cdot 10^4$ files. We aim to verify:

1. the scalability of CoBank in terms of the success rate and three major types of overhead.

2. the robustness of CoBank against node churn and malicious peers.

5.5.1.2 Network Model

The single hop Round-Trip-Time (RTT) in DHT routing is simulated by drawing 10^4 nodes from the Xbox 360 data set [127] that is spread over Western U.S. The mean, median, and standard deviation of inter-peer RTT of this data set are 81 ms, 64 ms, and 63 ms. We use the two-state Gilbert model [83], which models the packet loss property of Internet paths, setting loss rate to 1% and mean loss burst time to 100 ms as in [35]. The distribution of peer inter-arrival times follows a Weibull distribution, and the session length distribution (i.e., how long peers remain in the system each time they appear) follows a log-normal distribution as the study of churn in [188]. By default, all results shown are simulated with 20% probability of churn. When a peer re-enters the system, its account information remains the same as when it last left.

In terms of willingness to share, we simulate two types of peers as in [140]: 10% of the peers are content-rich, each sharing 100 files; 90% of the peers are content-lacking, each sharing 20 files. The file download request is generated at each peer following a Poisson process with average arrival rate one request per 2000 seconds. The price of downloading a file is 10, and each buyer randomly chooses a vendor if there are multiple vendors available. By default, the reward for good account holders $\$_{Bn}$ is 10 with the reward period $4 * 10^4$ seconds. The reward period is set to cover the time duration of 20 requests on average, so that the reward transaction overhead is balanced off. And the fine for bad account holders $\$_{Fn}$ is 100 followed the rules in [25].

We simulate the system for $7.2 * 10^5$ seconds in each simulation run and conduct over 20 runs for each result.

5.5.1.3 Performance Metrics

We adjust our timeout and DHT parameters to guarantee 100% successful request rate if the buyer has enough currency to pay. Then, we mainly present the results of three types of overhead when the success rate acheives 100%: (1) the *storage overhead* is defined as the average storage space at each node for storing account information or transaction information; (2) the *communication overhead* is defined as the average bandwidth consumption for each transaction; (3) the *latency overhead* is defined as the average delay from issuing a request to the completion of the transaction.

5.5.2 Evaluation Results

5.5.2.1 Performance impacts of the division factor

We increase *the division factor* k from 2 to 11, and measure the three types of overhead as shown in Figures 5.3, 5.4, 5.5. We compare the results when the replication factor is 1 (i.e., no replication) or 3 (i.e., minimum requirement for detecting malicious peers).

As each account is decomposed into k parts, each of which is stored at a different holder, the minimum requirement for preventing a holder from impersonation is $k = 2$. The storage overhead as in Figure 5.3 and communication overhead as in Figure 5.4 are linearly increasing with the division factor k. And the latency overhead slightly increases with k increases as in Figure 5.5. This is because if the network is error-free, the latency is kept the same when the network size is unchanged, but the non-zero packet loss rate causes some arbitrator to abort the transaction due to timeout. In this case the account holder needs to retransmit, and the latency overhead of that transaction increases.

We set the default value of the division factor k to 2 because larger k is only

useful for preventing account holder collusions but at the cost of increasing all three types of overhead. Since we use replication (i.e. the replication factor $m \geq 3$) to detect malicious holders, the division factor 2 suffices for prevent accounts being impersonated with low overhead.

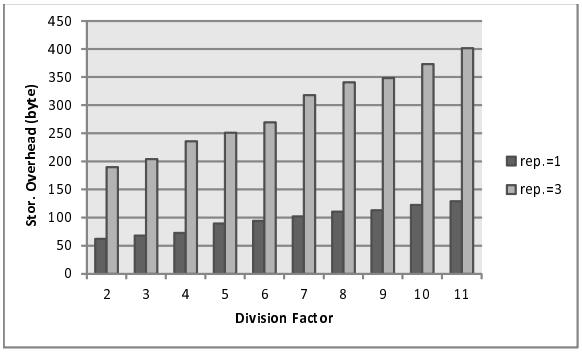

Figure 5.3: Storage overhead with varying division factors, where rep. – the replication factor

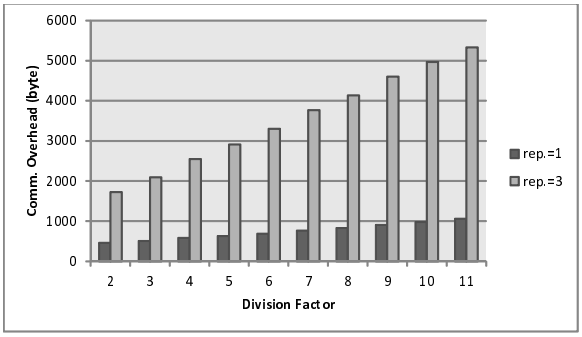

Figure 5.4: Communication overhead with varying division factors, where rep. – the replication factor

5.5.2.2 Performance impacts of the replication factor

We increase *the replication factor* m from 3 to 12, and measure the three kinds of overhead as shown in Figures 5.6, 5.7, 5.8. We compare the results when division factor is 2 (i.e., minimum requirement) or 4.

As each account is replicated and stored at m sets of account holders, the

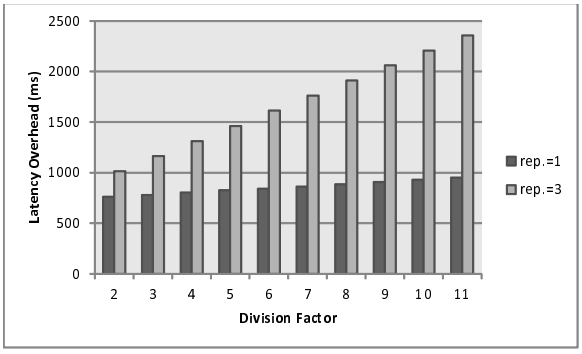

Figure 5.5: Latency overhead with varying division factors, where rep. – the replication factor

minimum requirement for detection and correction of any holder or arbitrator cheating is $m = 3$. The storage overhead as shown in Figure 5.6 is linearly increasing and the communication overhead as shown in Figure 5.7 is quadratically increasing with the replication factor m. The latency overhead as shown in Figure 5.8 increases with m. The reason is similarly to that with k, retransmissions may occur due to the timeout caused by packet loss.

We set the default value for the replication factor $m = 4$ to ensure our scheme with 100% successful request rate under node churn and with normal range of malicious peers. The reason is that the replication factor $m = 3$ can ensure nearly 100% successful DHT routing as long as churn rate is no more than 50% [170], but we need to ensure that at least requests from 3 sets of account holders are successfully received to detect malicious nodes. Hence, we choose $m = 4$ to protect the security with low overhead.

5.5.2.3 Comparisons in Overhead under Churn

We compare our scheme CoBank with PledgeRoute [125] because both are distributed indirect reciprocity schemes for P2P systems. The primary difference is that

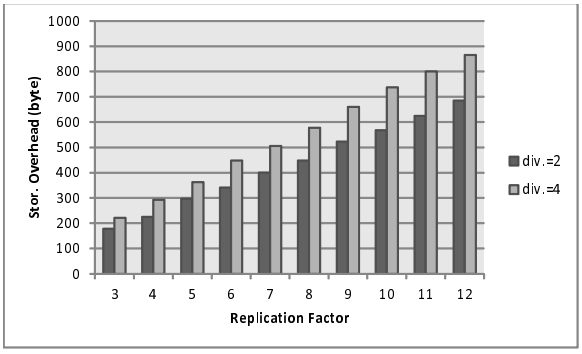

Figure 5.6: Storage overhead with varying replication factors, where div. – the division factor

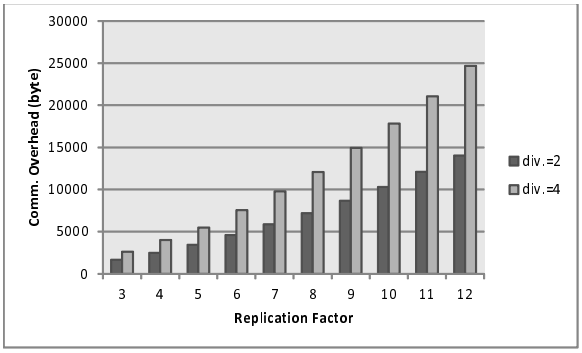

Figure 5.7: Communication overhead with varying replication factors, where div. – the division factor

PledgeRoute provides pairwise currency while CoBank provides global currency. We compare the three types of overhead in both schemes with varying the churn rate as in Figures 5.9, 5.10, 5.11.

For fairness, the results of overhead are measured where both schemes ensure almost 100% successful request rate if buyers have enough currency. Specifically, we set the parameters as follows: in CoBank the division factor k is 2, the replication factor m is 4, and the timeout T_{out} is 280ms, which is the average RTT from any two peers through DHT routing. In PledgeRoute, each peer has 150 neighbors on average, and 25 random-walk announcements are issued by each peer periodically with stopping probability 0.01 for topology discovery of the contribution network as in PledgeRoute [125]. The length of the announcement period is adjusted based on the churn rate to

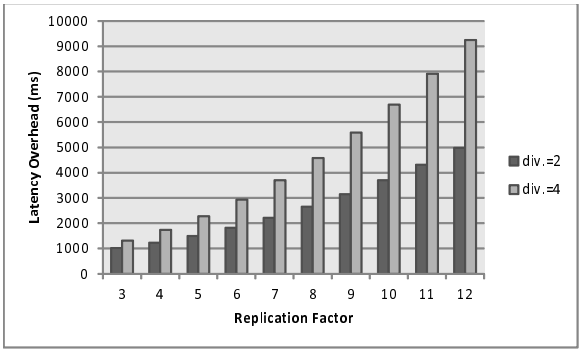

Figure 5.8: Latency overhead with varying replication factors, where div. – the division factor

ensure the success rate.

As shown in Figure 5.9, our storage overhead is an order of magnitude lower than that of PledgeRoute because PledgeRoute storage overhead is linear to the total number of nodes while ours is kept to a small constant. With increasing churn rates, the storage overhead of PledgeRoute increases faster than CoBank, as each PledgeRoute peer has to maintain a distinct account for every other peer who once appeared in the network. However, CoBank storage overhead is only based on our division factor k and replication factor m, independent of the total number of nodes.

Our communication overhead is only half of that in PledgeRoute as in Figure 5.10. It is because each node in PledgeRoute has extra topology discovery overhead for constructing their local contribution networks [125], which is counted into the average communication overhead for each transaction. The topology discovery frequency increases for higher churn rates. Moreover, for each transaction a PledgeRoute buyer has to find and reserve a valid contribution transfer path in its local contribution network. In the worst case, finding a path may have the communication overhead linear to the network size. However, we use global currency which is uniform in the network so that our peers do not have to know others' contributions. Each transaction overhead

only consists of the communication among the buyer/vendor, their account holders and arbitrators, which only grows logarithmly with the network size.

We incur latency overhead 1/3 less than PledgeRoute as in Figure 5.11 because for each transaction in PledgeRoute the buyer may fail several times in finding and reserving a valid contribution transfer path due to churn or lack of currency in the intermediate nodes. This adds to extra latency overhead in their scheme, which becomes larger with increasing churn. CoBank uses replications of account holders and arbitrators to combat churn with only slight latency increase.

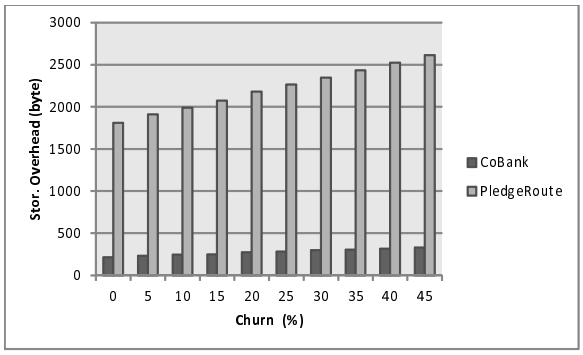

Figure 5.9: Storage overhead with varying churn

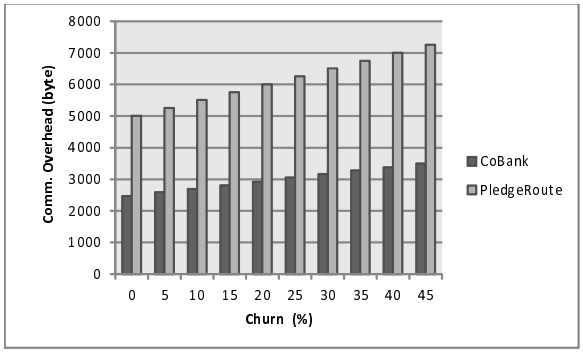

Figure 5.10: Communication overhead with varying churn

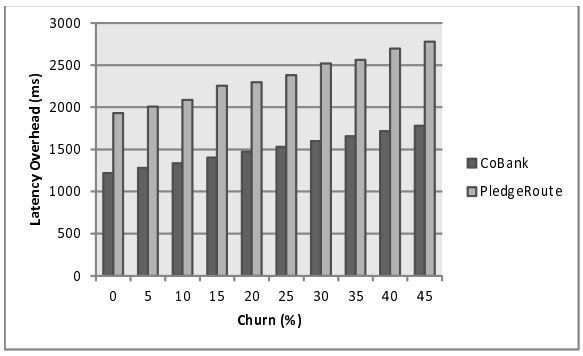

Figure 5.11: Latency overhead with varying churn

5.5.2.4 Comparisons in Robustness with Malicious Nodes

We simulate malicious nodes by having account holders and arbitrators cheating in CoBank, and nodes cheating in their contribution networks in PledgeRoute. We compare transaction success rates of two schemes by varying the percentage of malicious nodes as shown in Figure 5.12. Without malicious nodes, CoBank achieves almost 100% success rate and PledgeRoute has slightly less because a few transactions fail due to lack of valid contribution transfer path. CoBank maintains 90% success rate with 10% of malicious nodes and around 50% success rate with almost 1/3 malicious nodes. This is because the replication of account holders and arbitrators is very effective for detecting and correcting malicious behaviors as long as the majority nodes are good. The success rate of PledgeRoute drops to lower than 25% when 1/3 of nodes are malicious because they use random-walk to construct local contribution networks. One malicious node fails all random-walk searches it receives. There is no guarantee in PledgeRoute for successfully contructing local contribution networks or finding a contribution transfer path, when malicious nodes are present. Therefore, their transaction success rate drops quickly.

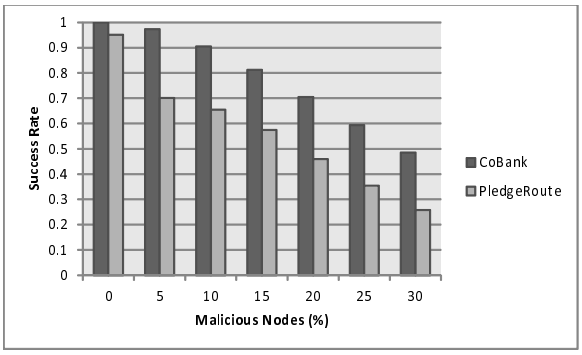

Figure 5.12: Success rate with malicious nodes

Chapter 6

Data Consistency Maintenance Framework in P2P Systems

6.1 Description of BCoM

BCoM aims to: (1) provide bounded consistency for maintaining a large number of replicas of a mutable object; (2) balance the consistency strictness, object availability for updating, and update propagation performance based on dynamic network conditions, workload patterns, and resource constraints; (3) make the consistency maintenance robust against frequent node churn and failures. To fulfill these objectives, BCoM organizes all replica nodes of an object into a d-ary dissemination tree (*dDT*) on top of the P2P overlay for disseminating updates. It applies three core techniques: the sliding window update protocol, the ancestor cache scheme, and the tree node migration policy on a *dDT* for consistency maintenance. In this section, we first introduce the *dDT* structure, and then explain the three techniques in detail.

6.1.1 Dissemination Tree Structure

For each object, BCoM builds a tree with node degree d rooted at the node whose ID is the closest to the object ID in the overlay identifier space. We denote this d-ary dissemination tree of object i as dDT_i. Each node in dDT_i is a peer who holds a copy of object i. We name such a peer as a "replica node" of i, or simply as a replica node. An update can be issued by any replica node, but it should be submitted to the root. The root serializes updates to eliminate conflicts.

With node churn and failures in P2P systems, a dDT serves two cases of insertions: (1) a single node joining, and (2) a node with subtree rejoining. The goal of constructing a dDT is to minimize the tree height with low overhead in both cases.

We show an example of dDT_i construction with node degree d set to 2 in Figure 6.1. The replica nodes are ordered by their arrival times as node 0, node 1, node 2, etc. At the beginning, node 1 and node 2 joined. Both were assigned by node 0 (i.e., the root) as its children. Then, node 3 joined. Since node 0 cannot have more child, it passed node 3 to a child who has the smallest number of subtree nodes. Since both children (i.e., node 1 and node 2) had the same number of subtree nodes, node 0 randomly selected one to break the tie, say node 1, and increased the number of subtree nodes at node 1 by one. Node 1 assigned node 3 as its child because it had a space for a new child. When node 4 joined, node 0 did not have space for a new child and passed node 4 to the child with the smallest number of subtree nodes, node 2. Similarly, node 5 and node 6 joined. When node 6 crashed, all of its children detected the crash independently and contacted other ancestors to rejoin the tree. Every child of node 6 acts as a delegate of its subtree to save individual rejoining of the subtree nodes. Section 6.1.3 explains how to contact an ancestor for rejoining. The tree construction algorithm

is given in Algorithm 1. We use $Sub_{no.}(x)$ to count the number of subtree nodes of node x, including itself.

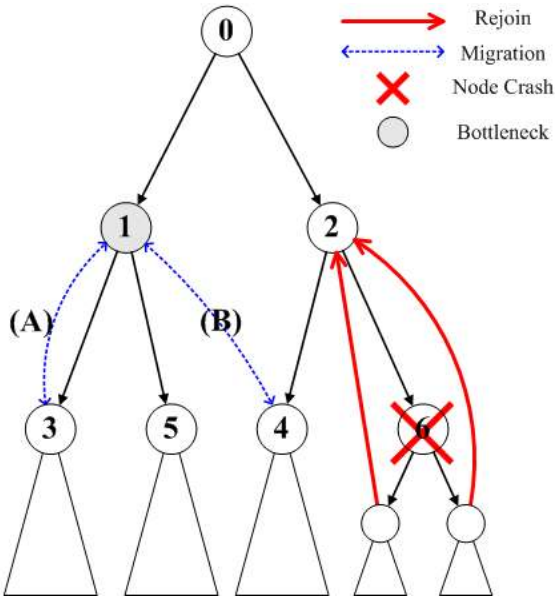

Figure 6.1: Dissemination Tree Example

Algorithm 1 (Dissemination Tree Construction)

Input: node p receives node q's join request

Output: parent of node q in dDT

IF p does not have d children

$Sub_{no.}(p) += Sub_{no.}(q)$

RETURN p

ELSE

find a child f of p s.t. f has the smallest $Sub_{no.}$

$Sub_{no.}(f) += Sub_{no.}(q)$

RETURN dDT Construction (f, q)

The *dDT* construction algorithm uses the number of subtree nodes as the met-

ric for insertions, instead of the tree depth used in traditional balanced tree algorithms. This is because a rejoining node with a subtree may increase the tree depth by more than one, which is beyond the one by one tree height increase handled by traditional balanced tree algorithms. In addition, maintaining the total number of nodes in each subtree is simpler and more time efficient than maintaining the depth of each subtree. Internal nodes need to wait until an insertion completes, then the updated tree depth can be collected layer by layer from leaf nodes back to the root. This makes the real time maintenance of the tree depth difficult and unnecessary when tree nodes are frequently joining and leaving. However, internal nodes can immediately update the total number of subtree nodes after forwarding a new node to a child. In BCoM, the tree depth is periodically collected to help set the sliding window size, where its result does not need to be updated in real time as discussed in Section 6.1.2.2. But using an outdated tree depth for insertions to dDT will lead to an unbalanced tree and degrade the performance.

6.1.2 Sliding Window Update Protocol

6.1.2.1 Basic Operations in Sliding Window Update

The sliding window update protocol regulates the consistency strictness in a dDT. "Sliding" refers to the incremental adjustment of the window size based on dynamic system conditions. If dDT_i of object i is assigned a sliding window size k_i, any replica node in dDT_i can buffer up to k_i unacknowledged updates before getting blocked from receiving new updates. In other words, each node in dDT_i is given a buffer of size k_i. At the beginning, the root receives the first update, sends it to all children and waits for their ACKs. There are two types of ACKs, R_ACK and NR_ACK. Both

indicate that the update has been received. The difference is that R_ACK means the sender is ready to receive the next update; NR_ACK means the sender is not ready. While waiting, the root accepts and buffers the incoming updates as long as its buffer is not overflowed. When receiving an R_ACK from a child, the root sends the next update to this child if there is a buffered update that has not been sent to this child. When receiving an NR_ACK from a child, it marks the update to be received by this child and stops sending update to this child. After receiving ACKs from all children, the update is removed from the root's buffer.

There are two cases of buffer overflow: 1) when the root's buffer is full, the new updates are discarded until there is a space; 2) when an internal node's buffer is full, the node sends NR_ACK to its parent for the last received update. An R_ACK is sent to its parent when the internal node has a space in its buffer to resume receiving updates. A leaf node does not have any buffer. After receiving an update, it immediately sends an R_ACK to its parent.

Figure 6.2 shows an example of the sliding window update protocol with the window size set to 8. V stands for the version number of an update, as $V10 - V13$ means that the node keeps the updates from the 10th version to the 13th version. Each internal node keeps the next version for its slowest child up to the latest version it received. Each leaf node only keeps the latest version it received.

6.1.2.2 Window Size Setting

The sliding window size k_i is critical for balancing the consistency strictness, object availability for updating, and update dissemination latency. A large k_i masks the

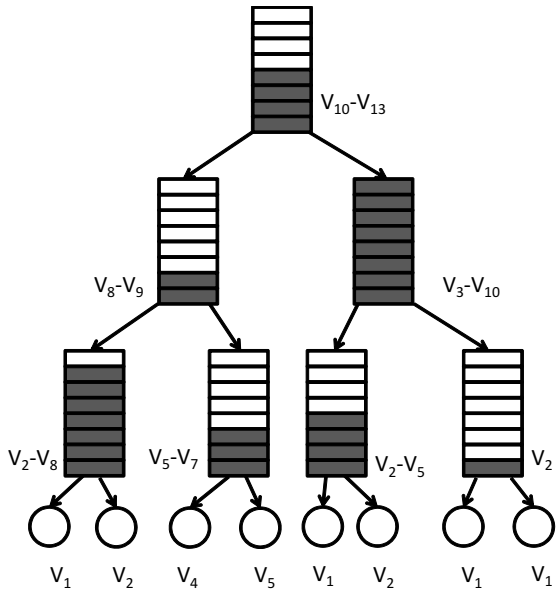

Figure 6.2: An example of sliding window update protocol.

long network latency and the temporary unavailability of replica nodes, thus lowers the update discard rate. But a large k_i enlarges the discrepancy between the local version of a replica node with the latest version at the root. Thus, a large window size k_i weakens the consistency and increases the queueing delay of update propagation in dDT_i. *On the extremes, infinite buffer size provides eventual consistency without discarding updates, and buffer size zero provides sequential consistency with the worst update discard rate.*

We present an analytical model in Section 6.2 to set the sliding window size k_i so that the discard rate is minimized under a delay constraint and a consistency constraint. The detail formula is given in Section 6.2. Here, we explain the procedure for setting the window size. The root sets the window size for all tree nodes and adjusts it periodically when needed. The root measures the input metrics for computing the window size every T seconds and adjusts the value of k_i only after the metrics stabilize and the old k_i violates certain constraints. In this way, unnecessary changes due to temporary disturbances are eliminated to keep dDT_i stable. If k_i needs to be adjusted,

it is incrementally increased or decreased until the constraints are satisfied.

The input metrics of computing the window size k_i include the update arrival rate λ, the tree height L, and the bottleneck service time μ_L. The arrival rate is directly measured by the root. The tree height and bottleneck service time are collected periodically from leaf nodes to the root in a bottom-up approach. The values of the two metrics are aggregated at every internal node so that the maintenance message keeps the same size. The aggregation is performed as follows: each leaf node initializes the tree height to zero ($L = 0$) and the bottleneck service time μ_L to the update propagation delay between itself and its parent. Each node sends the maintenance message to its parent. Once an internal node receives the maintenance messages from all children, it updates L as the maximum value of its children's tree height plus 1 and μ_L as the maximum value among its and every child's service time. If its service time is longer than a child's, a non-blocking migration is executed to swap the parent with the child. The aggregation continues until the root is reached.

6.1.3 Ancestor Cache Maintenance

Each replica node maintains a cache of m_i ancestors starting from its parent leading to the root in dDT_i. The value of m_i is set based on the node churn rate (i.e., the number of nodes joining and leaving the system during a given period) so that the probability that all m_i nodes simultaneously fail is negligibly small. If a node does not have m_i ancestors, it caches all the ancestors from its parent to the root.

A node contacts its cached ancestors sequentially layer by layer upwards when its parent becomes unreachable. This can be detected by ACKs and maintenance messages. Sequentially contacting enables a node to find the closest ancestor. The root is finally contacted if all the other ancestors are unavailable. The root failure is handled

by the overlay routing, as a node with the nearest ID will replace the crashed node to be the new root. Different replication schemes may be used to reduce the cost of root failure, which is specific to a structured P2P overlay and beyond the scope of this paper.

The contacted ancestor runs the tree construction Algorithm 1 to find a new position for a rejoining node with its subtree. BCoM does not replace a crashed node with a leaf node to maintain the original tree structure because migration brings down a bottleneck node to the leaf layer for performance improvement. The new parent node transfers the latest version of the object to the new child if necessary. Since each node only keeps k_i previous updates, content transmission is used to avoid the communication overhead for getting the missing updates from other nodes. The sliding window update protocol resumes for incoming updates.

The ancestor cache provides fast recovery from node failures with a small overhead. Assuming the probability of a replica node failure is p, an ancestor cache of size m_i has a successful recovery probability as $1 - p^{m_i}$. An ancestor cache is easily maintained by piggybacking an ancestor list on each update. Whenever a node receives an update it adds itself to the ancestor list before propagating the update to its children. Each node uses the newly received ancestor list to refresh its cache. There is no extra communication, and the storage overhead is also negligible for keeping the information of m_i ancestors.

6.1.4 Tree Node Migration

Any non-leaf node will be blocked from receiving new updates if one of its descendants has a buffer overflow in the sliding window update protocol. It is quite possible that a lower layer node performs faster than a bottleneck node. This motivates us to promote the faster node to a higher level and degrade the bottleneck node to a

lower level. For example in Figure 6.1, assume node 1 is the bottleneck node causing the root 0 to be blocked. The faster node may be a descendant of the bottleneck node as shown in (A) or a descendant of a sibling of the bottleneck node as shown in (B). When blocking occurs, node 0 can swap the bottleneck node 1 with a faster descendant who has more recent updates, like node 4, to remove blocking. Before blocking occurs, node 1 can be swapped with its fastest child who has the same update version to prevent blocking. The performance improvement through node migration is confirmed by our analytical model in Section 6.2.

There are two forms of node migration, as described below.

- Blocking triggered migration: the blocked node searches for a faster descendant who has a more recent update than the bottleneck node, and swaps them to remove blocking.

- Non-blocking migration: when a node observes a child performing faster than itself, it swaps with this child. Such migration prevents blocking and speeds up the update propagation for the subtree rooted at the parent node.

The swapping of (B) in Figure 6.1 is an example of blocking triggered migration and (A) is an example of non-blocking migration. Both forms of migration swap one layer at a time and, hence, multiple migrations are needed for multi-layer swapping. The non-blocking migration helps promote faster nodes to upper layers, which makes the searching in blocking-triggered migration easier. Since the overlay DHT routing in structured P2P networks relies on cooperative nodes, we assume BCoM is run by these cooperative P2P nodes transparent to end users. Tree node migration uses only the local information and improves the overall system performance.

6.1.5 Basic Operations in BCoM

BCoM provides three basic operations:

- Subscribe: if a node p wants to read the object i and keep it updated, p sends a subscription request to the root of dDT_i through the overlay routing. After receiving the request, the root runsAlgorithm 1 to locate a parent for p in dDT_i, who will transfer its most updated version of object i to p. p receives the subsequent updates by following the sliding window update protocol. The message overhead of a subscription is at most the tree height as inserting a new node searches along a path from the root to a leaf in dDT_i.

- Unsubscribe: if a node p is not interested in object i anymore, it promotes its fastest child as the new parent and transfers its parent and other children's information to the newly promoted node. p also notifies them of the newly promoted node to update their related maintenance information. The message overhead of an unsubscription is constant, since the number of involved nodes is no more than the tree node degree, and each node has a constant overhead to update its local maintenance information.

- Update: after subscribing, if a node p wants to update the object, it sends an update request directly to the root using the IP routing. The root's IP address is obtained through the subscription or the ancestor cache. If the root crashed, p submits the update to the new root through the overlay routing. Updates are serialized at the root by their arrival times. The specific policy for resolving conflicts is application dependent. The message overhead of an update is constant for the direct submission to the root.

6.2 Analytical Model for Sliding Window Setting

The frequent node churn in P2P systems forbids us to use any complicated optimization techniques that require several hours of computation at workstations (e.g., [232]) or every node information in the system (e.g., [235]). BCoM adjusts the sliding window size to the dynamic P2P systems relying on limited information.

This section analyzes the setting of the sliding window size k_i for object i, where the update propagation to all replica nodes is modeled by a queuing system. We first analyze the queueing behavior of the dissemination tree dDT_i when it begins to discard an update. We then calculate the update discard probability and the expected latency for a replica node to receive an update. Finally, we set k_i to minimize the update discard rate given a consistency bound while ensuring the expected delay is no worse than the baseline by a small given threshold.

6.2.1 Queueing Model

Assuming the total number of replica nodes is N, the node degree is d, and there are L ($L = O(\log_d N)$) layers of internal nodes with an update buffer of size k_i (i.e., each node in layer $0 \dots L-1$ has a sliding window k_i). The leaf nodes are in layer L and do not have any buffer. The update arrivals are modeled by a Poisson process with an average arrival rate λ_i (simply as λ), since each update is issued by a replica node independently and identically at random. The latency of receiving an update from the parent and an acknowledgment from the child is denoted as the service time for an update propagation. The service time for one layer to its adjacent layer below is the longest parent-child service time in these two layers. μ_l denotes the service time for update propagation from layer l to layer $(l+1)$. For example, μ_0 is the longest service

time from the root to its child, μ_{L-1} is the longest service time from a layer $(L-1)$ node to its child (i.e., a leaf node). The update propagation delay is assumed to be exponentially distributed. The update propagation in dDT_i is modeled as a queuing process as shown in Figure 6.3 (a): updates arrive at the root with an average rate λ, then go to the layer 0 node's buffer of size k_i. The service time for propagating from layer 0 to layer 1 is μ_0. After that, the updates go to a layer 1 node's buffer of size k_i with service time μ_1 for propagating to a layer 2 node. The propagations end when updates are received by a leaf node in layer L.

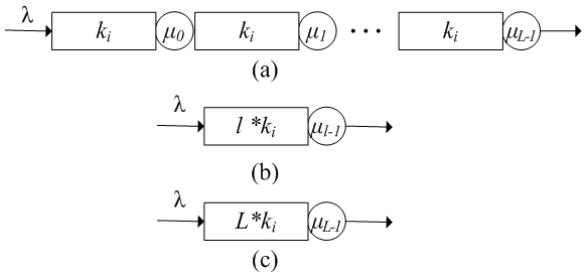

Figure 6.3: Queuing Model of Update Propagation.

An update may only be discarded by the root when its buffer is overflowed. This happens when the root is waiting for an R_ACK from its slowest child in layer 1, who is waiting for an R_ACK from its slowest child in layer 2. The waiting cascades until a bottleneck node of dDT_i is reached, say in the layer l, $0 \leq l \leq L$. The nodes in layers $l+1 \ldots L$ (if $l < L$) do not receive any update even when their buffers are not full. All the nodes in the path from the root to the bottleneck node have buffer overflow. The nodes along the path are denoted as $p_0, p_1 \ldots p_l$, where p_0 is the root and p_l is the bottleneck node. After the bottleneck node p_l has a space in its buffer and sends an R_ACK to its parent, the R_ACK is then propagated to the root p_0 such that the root can purge the update from its buffer and accept a new one. The update

propagation from $p_0 \rightarrow p_1, p_1 \rightarrow p_2, \ldots p_{l-1} \rightarrow p_l$ is in parallel and the service time μ_{l-1} between p_l and p_{l-1} should be the longest along this path (i.e., $\mu_{l-1} > \mu_j, 0 \leq j < l-1$). Therefore, the queuing model of the update discard is transformed to a queue with an effective buffer of size $l * k_i$ for dDT_i, and the service time is μ_{l-1}, as shown in Figure 6.3 (b).

This queuing model explains that given a k_i, the effective buffer size $l * k_i$ is determined by l, which is the layer of the bottleneck node. The larger the effective buffer size, the lower the discard probability. When the bottleneck node is a leaf node ($l = L$), buffer resources of dDT_i are fully used with an effective buffer size $L * k_i$. This inspires the Tree Node Migration techniques presented in Section 6.1.4, which moves down bottleneck nodes to the leaf layer. The discard probability of an update is computed based on the queuing model of dDT_i after being optimized by tree node migrations as shown in Figure 6.3 (c). The queue becomes an $M/M/1/$ queue with a buffer size $L * k_i$, an arrival rate λ and a service time μ_{L-1}.

6.2.2 Availability and Latency Computation

Define the update request intensity as ρ.

$$\rho = \frac{\lambda}{\mu_{L-1}} \tag{6.1}$$

Define the probability of n updates in the queue as π_n. Based on the queueing theory for $M/M/1$ finite queue [33], π_n is represented as Equation 6.2.

$$\pi_n = \rho^n \pi_0 \tag{6.2}$$

The discard probability is π_{L*k_i}, which indicates the buffer overflow. From $\Sigma_{n=0}^{L*k_i} \pi_n = 1$, we get $\pi_0 = \frac{1-\rho}{1-\rho^{L*k_i}}$. And the discard probability is computed in Equation 6.3.

$$\pi_{L*k_i} = \frac{1-\rho}{1-\rho^{L*k_i}} \rho^{L*k_i} \tag{6.3}$$

The expected number of packets in the queue $E[N_{L*k_i}]$ is calculated in Equation 6.4.

$$E[N_{L*k_i}] = \sum_{0 \leq n \leq L*k_i} n * \pi_n \tag{6.4}$$

Plug in the Equation 6.2 for π_n, the final form of $E[N_{L*k_i}]$ is given in Equation 6.5.

$$E[N_{L*k_i}] = \frac{(L*k_i+1)\rho^{L*k_i+1}}{(\rho^{L*k_i+1}-1)} + \frac{\rho}{(1-\rho)} \tag{6.5}$$

The expected delay $E[T_{L*k_i}]$ is calculated by Little's law in Equation 6.6, where $E[N_{L*k_i}]$ is the expected number of packets in the queue and $\lambda(1-\pi_{L*k_i})$ is the arrival rate of the accepted updates.

$$E[T_{L*k_i}] = \frac{E[N_{L*k_i}]}{\lambda(1-\pi_{L*k_i})} \tag{6.6}$$

6.2.3 Window Size Setting

The effectiveness of a consistency protocol is measured by three attributes: consistency strictness, object availability, and latency for receiving an update. The three are in subtle tension towards each other. Given the update arrival rate and the service time, increasing the window size k_i lowers the discard probability, while prolongs the expected latency and weakens the consistency strictness. π_{L*k_i} is the discard probability. The expected latency $E[T_{L*k_i}]$ indicates the average delay for an update to be received by a replica node. The consistency strictness is measured by the number of updates a replica node has not yet received, which is at most $L * k_i$ in dDT_i.

BCoM sets the window size to minimize the update discard rate under the constraints that the number of not-yet-received updates is bounded to K_m and the latency for receiving an update is no worse than the sequential consistency for a small bound T_s as calculated in Equation 6.7. $E[T_{L*k}]$ is the expected latency with a window size k and $E[T_L]$ is the expected latency when applying sequential consistency to dDT_i, which

serves as the baseline for bounding the latency performance. The latency threshold T_s and the consistency strictness threshold K_m are set according to application requirements. In our simulation, empirically setting T_s to 1.3 achieves good results as shown in Figure 6.9 and Figure 6.11, the discard probability is improved from almost 100% to 5% at the cost of latency increases less than one third most of the time. K_m is set to 60 for a network of 1000 nodes.

$$k_i = arg\min\ \pi_{L*k}\ s.t.\ \frac{E[T_{L*k}]}{E[T_L]} \leq T_s, L*k \leq K_m \tag{6.7}$$

6.3 Performance Evaluation

In this section, we extend the P2PSim tool [12] to simulate BCoM with heterogeneous node capacities and transmission latencies. While BCoM can be applied to any type of structured P2P systems, we choose Tapestry[238] as a representative network for simulations. We evaluate the efficiency of BCoM with comparison to SCOPE[55], which is the most relevant work and a widely studied consistency technique in structured P2P systems.

6.3.1 Simulation Setting

We simulate a network of 1000 nodes because anything larger cannot be executed stably in P2PSim. The number of objects ranges from 10^2 to 10^4. The object popularity follows a Zipf's distribution, and the update arrivals are generated by a Poisson process with different average arrival rates. By default, each node issues 200 updates during a simulation cycle, which is $7.2 * 10^6$ time slots. We simulate the situation where frequent updates may overload the servers, which motivates the use of P2P systems. Given that transmitting one update uses only 10 to 100 time slots, the number of time

slots covered in a simulation cycle (i.e., $7.2 * 10^6$) is large enough to generate sustainable results. The data points in our figures are the average values of 20 trials.

The heterogeneity of node capacities follows a Pareto distribution [136]. We set the shape parameter $a = 1$ and scale parameter $b = 900$ to get 900 different node capacities. Network topology is simulated by two transit-stub topologies generated by GT_ITM [73] to model dense and sparse networks: (1) ts1k-small (dense) - 2 transit domains each with 4 transit nodes, 4 stub domains attached to each transit node, and 31 nodes in each stub domain. (2) ts1k-large (sparse) - 30 transit domains each with 4 transit nodes, 4 stub domains attached to each transit node, and 2 nodes in each stub domain.

The node degree is set to 5 based on the average Gnutella node degree, which is 3 to 5. To have a fair comparison, we also set the vector degree of each SCOPE node to 5. The *update discard rate* (the ratio of the number of discarded updates to the total number of updates), and the *update dissemination latency* (the average delay for a replica node to receive an update) are used to measure the performance.

6.3.2 Efficiency of the Window Size

This simulation examines the efficiency of applying sliding window update protocol. The curves in Figure 6.4 and Figure 6.5 show that by increasing the window size from 1 to 20, the discard rate is dropped from 80% to around 5%, and the latency is increased only by 20%. The results confirm that BCoM significantly improves the object availability with slightly increased latency compared to applying the sequential consistency.

Impacts of the Window Size on the Extent of Inconsistency: This

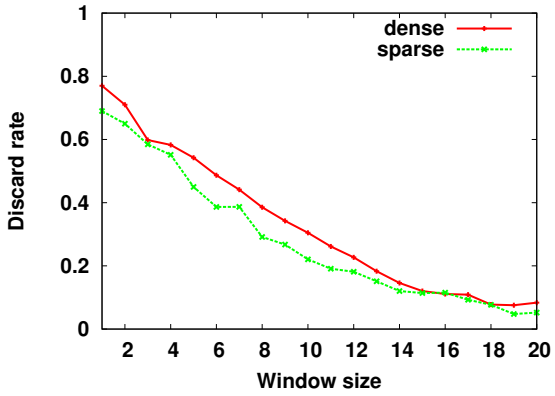

Figure 6.4: The impact of window size on discard rate.

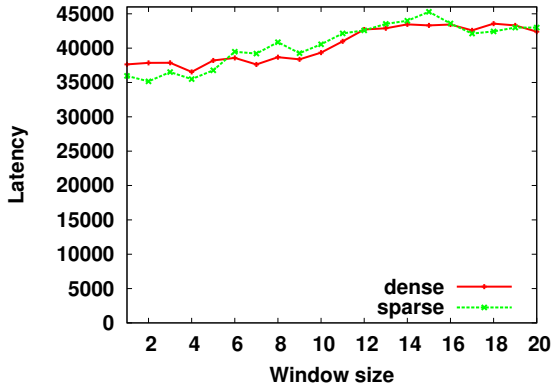

Figure 6.5: The impact of window size on latency.

simulation examines the extent of inconsistency among all replicas of an object by varying the window size. Figure 6.6 shows the results, where the inconsistency extent is defined as the average number of updates a replica falls behind the latest version at the root. Since the upper bound of inconsistency in BCoM is the window size multiplied by the tree height of a dDT, we use the average to show the real situation instead of the theoretical upper bound. As expected, the inconsistency extent grows larger as the window size increases, however, it grows much slower than the upper bound. When the window size is 20 and the tree height is around 4 to 6, the upper bound is around 80 to 120. But the average inconsistency extent is only 11, much smaller than the upper

bound. Given the total number of replica nodes is 1000, such inconsistency is quite acceptable when the update discard rate is dropped to only 5% as shown in Figure 6.4.

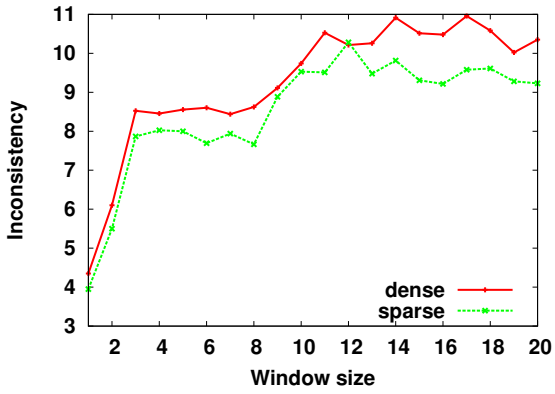

Figure 6.6: The impact of window size on inconsistency degree.

Accuracy of the Analytical Model: This simulation examines the accuracy of the analytical model for window size setting presented in Section 6.2. We compare the latency estimated by Equation 6.6 with the simulation latency results as shown in Figure 6.5. We also show the error rate of latency estimation in Figure 6.7, which is the ratio of the difference between the estimation and the simulation results over the simulation results. The discrepancy is mainly caused by the node churn because node leavings and rejoinings introduce extra delay and change parts of the tree structure. However, the error rates are less than 25% with different window sizes. Such small error rates indicate that our analytical model has captured the queueing behavior of the update dissemination in BCoM, which is the dominant factor to the system performance.

Storage Overhead: This simulation shows the storage overhead for buffering updates. The upper bound of storage overhead at a replica is the update packet size multiplied by the window size (i.e., the maximum buffer size). Figure 6.8 presents the

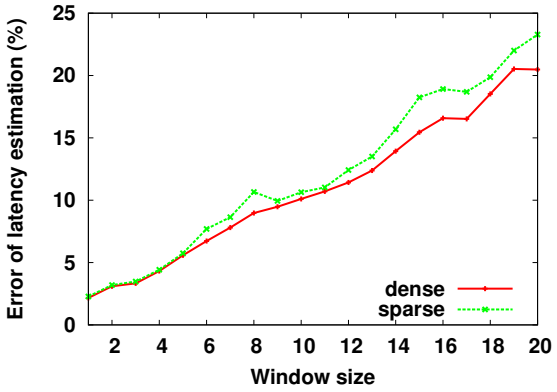

Figure 6.7: The impact of window size on latency estimation.

average buffer occupancy of all replicas to show the storage overhead in the real situation instead of the theoretical upper bound. The average buffer occupancy decreases as the window size increases, when the window size is 20 the average buffer occupancy is only 10%. Thus, most replicas have small storage overhead. This is because the window size is determined by the bottleneck service time (i.e., the slowest replica's service time) to reduce the update discard rate, other replicas only have a small number of udpates buffered even when the window size is large.

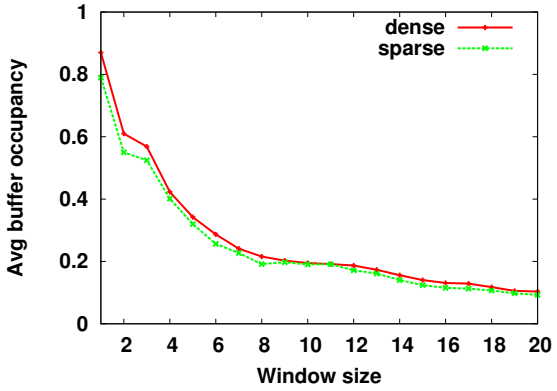

Figure 6.8: The impact of window size on storage overhead.

6.3.3 Scalability of BCoM

This simulation verifies the scalability of BCoM with comparison to SCOPE[55] by varying the number of replica nodes and the update rate of each object. The results in Figure 6.9 and Figure 6.10 show that the discard rate of BCoM is maintained to less than 10% as the number of replicas per object increases from 10 to 1000 and the number of updates issued per node increases from 1 to 200. On the other hand, applying the sequential consistency makes the discard rate of SCOPE almost 100%, except for a very small number of replica nodes (i.e., 10 replicas per object) or an extremely low update rate (i.e., 1 update per node). An update cannot be accepted by SCOPE until the previous update is received by every replica node. The prohibitively long synchronization for the sequential consistency makes SCOPE discard most updates.

We intentionally relax the consistency requirement for SCOPE when calculating their discard rate, latency and overhead results by not requiring their new joining/rejoining nodes to be synchronized. This relaxation gives better results to SCOPE for all three metrics. Without this relaxation, SCOPE's discard rate is even worse, which is not useful for comparison. Thus, the results of BCoM include content transfer delay for all joining/rejoining nodes while the results of SCOPE exclude the synchronization delay for all joining/rejoining nodes. This is an important reason why BCoM has slightly longer dissemination latency than that of SCOPE when the number of replica nodes is large as shown in Figure 6.11 or the updates are frequent as shown in Figure 6.12.

Another critical reason why SCOPE has slightly better latency and overhead is that it discards a large portion of updates. The measurements of latency and overhead only count accepted updates. With such high discard rate, SCOPE takes advantage of

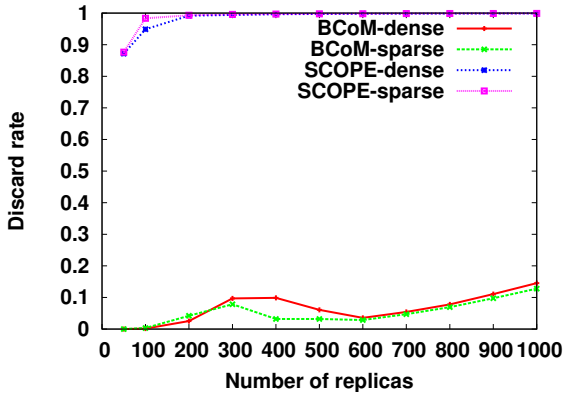

Figure 6.9: The impact of replica number on discard rate.

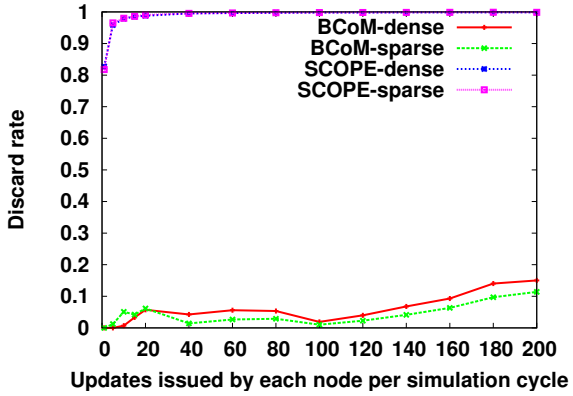

Figure 6.10: The impact of update pattern on discard rate.

accepting much fewer updates than BCoM when measuring latency and overhead.

6.3.4 The Overhead of BCoM

This simulation compares the overhead of BCoM with that of SCOPE as shown in Figure 6.13. The consistency maintenance overhead of each object consists of three parts: subscription overhead, update overhead, and crash/migration overhead. We use the label "migrate" to indicate the migration and crash recovery overhead in BCoM. BCoM keeps the overhead at the same level as that in SCOPE because the ancestor

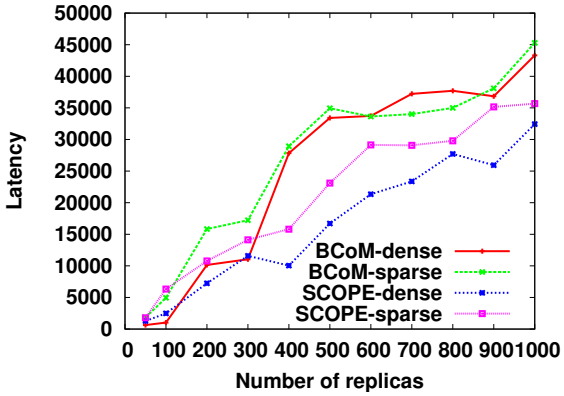

Figure 6.11: The impact of replica number on latency.

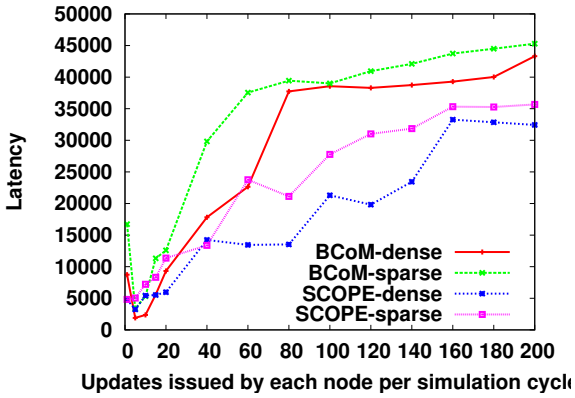

Figure 6.12: The impact of update pattern on latency.

cache maintenance and the node migration mostly piggyback on update dissemination.

subsectionFault Tolerance of BCoM This simulation examines BCoM's robustness against node failures by varying the node mean life time. The node life time is the ratio of the average number of slots a node stays online at one time to the total number of slots in a simulation cycle. The smaller the life time is, the more frequently the nodes join and leave. The results of SCOPE are not presented because their discard rate is nearly 100% in the presence of nodes joining and leaving. Figure 6.14, Figure

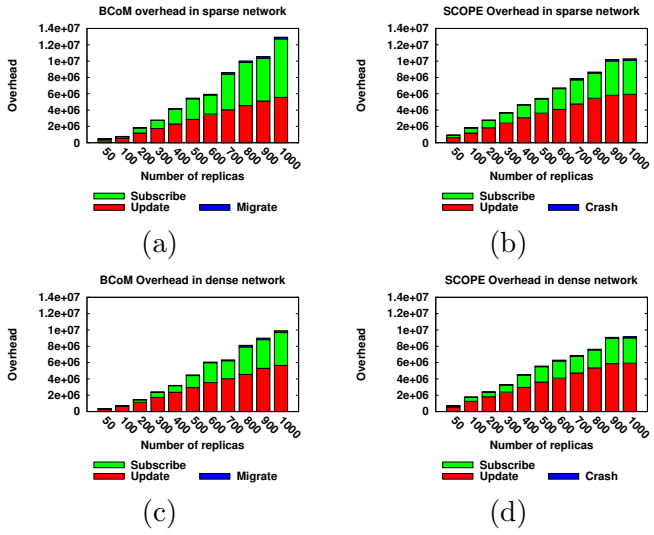

Figure 6.13: Overhead comparison between BCoM and SCOPE

6.15, and Figure 6.16 show the impacts of various churn rates on the dDT tree height, the update discard rate, and the update dissemination latency. The results demonstrate that BCoM is robust against the node churn. Figure 6.14 shows that the tree height is ranging from 4 to 6, which means our dDT is nearly complete as a 5-ary tree of 1000 nodes has tree height more than 4. The discrepancy caused by node churn on the tree height is less than 2, which helps BCoM maintain consistent discard rate and latency. Buffering some earlier updates in each intermediate node masks the delay for nodes to rejoin/join the tree and keeps the discard rate low under the node churn as shown in Figure 6.15. The efficient dDT construction and the use of ancestor cache reduce the delay for node joining/rejoining and prevent the degradation of update dissemination as shown in Figure 6.16. Note that the latency goes down sharply when node life time is small. The reason is that a node clears its buffer when it goes offline. Therefore, with an extremely short life time, a node's buffer is always near empty so that the queueing delay in this case is also extremely short.

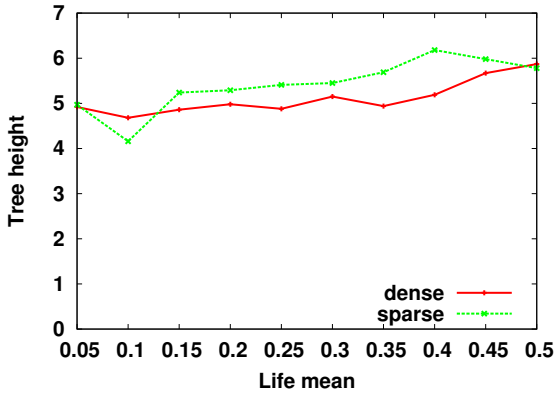

Figure 6.14: The impact of churn rate on tree height

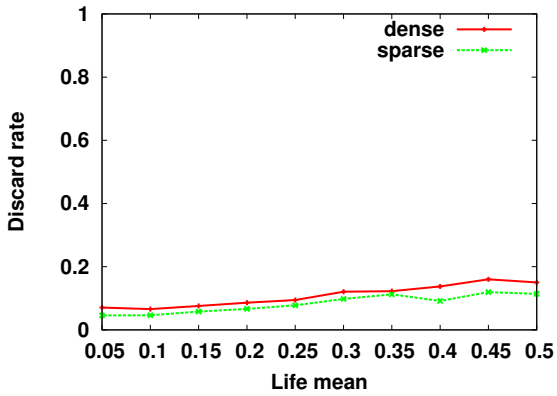

Figure 6.15: The impact of churn rate on discard rate

6.4 Case Study

In this section, we evaluate the performance of BCoM using a large-scale social networking application FriendFeed [49]. FriendFeed is a real-time feed aggregator that consolidates the updates from social media and social networking websites. It is created in 2007 and acquired by Facebook in 2009. Existing solutions to implement social networking is using dedicated servers, which are notoriously difficult to scale. Social network applications are incessantly evolving as more new users join with more frequent social interactions. Servers' capacities should continuously keep up for the growing

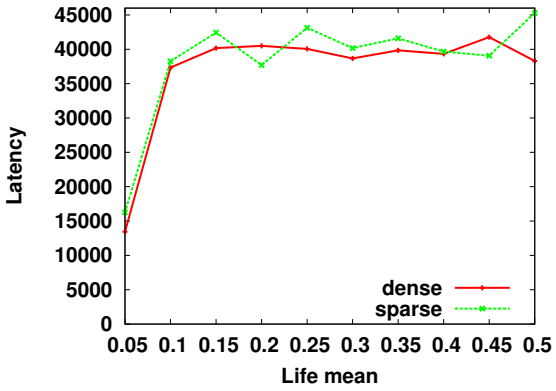

Figure 6.16: The impact of churn rate on latency

demand. Twitter engineers have famously described re-architecting Twitters servers multiple times to keep up with rapid increases in throughput as the system became more popular [3]. We believe that the P2P approach is the direction for the future social network applications with better scalability. Moreover, the P2P implementation is economic-friendly, which is particularly appealing to start-ups as we envision new social network applications are emerging.

6.4.1 Trace Data and Experimental Setup

Friendfeed users share posts on his/her blog with a list of subscribers, who can comment directly under the original blog post. BCoM is applied to FriendFeed for maintaining consistency between a FriendFeed user and all his/her subscribers. Each user's blog is modeled as an object in BCoM. Either a post or a comment is an update about the object, and a subscriber to a FriendFeed user is a replica node of the object (i.e., the user's blog).

Workload Model. We use the real trace data in [49] to generate the workload of subscriptions and updates. The trace includes 671840 FriendFeed users, and is collected from Aug 1, 2010 to Sep 30, 2010. Table 6.1 gives the summary of the trace

Table 6.1: Summary of FriendFeed Traces

Total nodes	671,840
Time duration	2 months
Total updates	16,200,549
Average replica nodes per object	41.40
Average updates per second	3.07
Average update packet size	354.91 bytes

data.

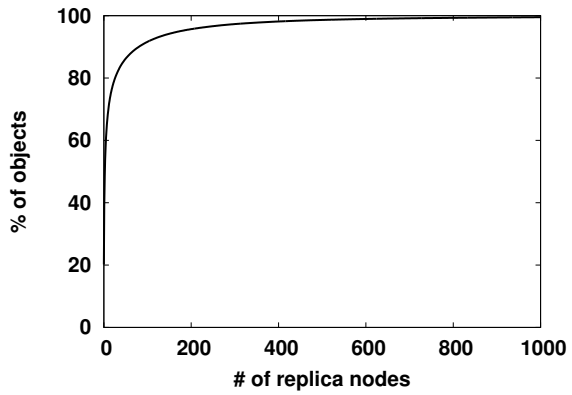

Figure 6.17: CDF of the number of replica nodes per object

We examine the main features of the trace data in Figures 6.17, 6.18, 6.19. Figure 6.17 shows the cumulative distribution function (CDF) of the number of replica nodes per object. The object popularity is highly skewed. More than 90% of objects have replica nodes less than 100, yet around 1% of objects have replica nodes more than 800. The maximum number of replica nodes per object reaches 113923. Figure 6.18 shows the CDF of the number of updates per object. The update distribution is also highly skewed. More than 90% of the objects have updates less than 50, while, around 2% of objects have updates more than 200. Figure 6.19 shows the total number of updates generated in the system, where we can see the udpate generation is unpredictable with

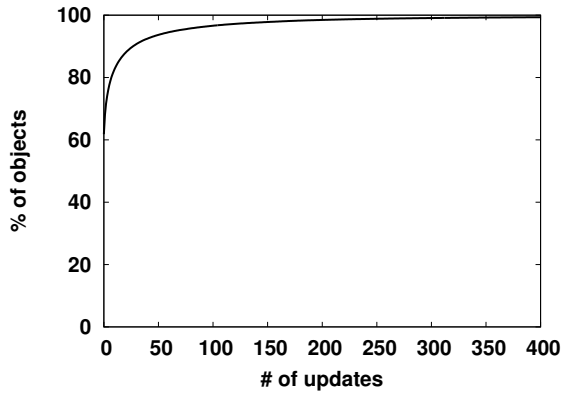

Figure 6.18: CDF of the number of updates per object

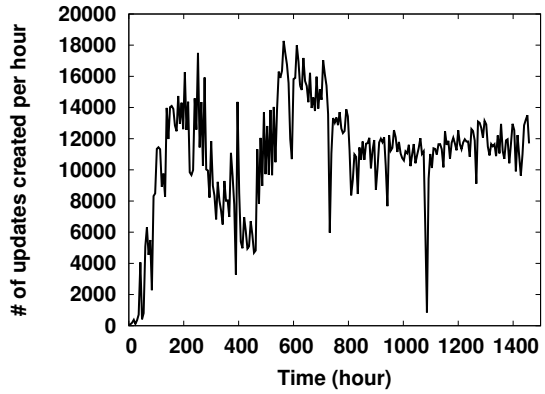

Figure 6.19: The total number of updates in the system per hour

frequent bursts. These trace analysis results show that the workload of update delivery and consistency maintenance is highly skewed among objects and constantly changing over time. One of the goals of BCoM is to provide balanced consistency maintenance for such workload patterns through our sliding window update protocol.

6.4.2 Network Model

. In our experiments, each peer is an individual machine in a simulated network, representing a general Internet user experience. To measure the update dissemination latency, we adopt the widely used statistics of the user bandwidth capacity collected

at U.S. Broadband report [2]. The upload capacity of peers is shown in Figure 7.6, which is in a range from 256 Kb/s to 10 Mb/s. To simulate wide geographic areas where peers come from, the inter-peer round-trip time (RTT) is simulated by drawing n ($n = 671840$ in our experiments) nodes from the real data set [127] which represents a wide area interactive application. The mean, median, and standard deviation of inter-player RTT of this data set are 81 ms, 64 ms, and 63 ms. Vivaldi 3D coordination system [61] is used to extrapolate the RTT values among pairs of nodes who did not probe each other in the data set. We use a two-state Gilbert model [83], which models the packet loss property of Internet paths, setting loss rate to 1% and mean loss burst time to 100 ms. The churn probability is set to 20% according to churn studies in [188]. The update dissemination tree structure in BCoM is configured the same as in our simulation in Section 7.4.

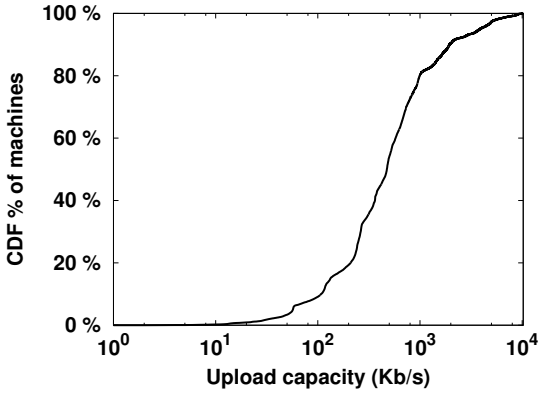

Figure 6.20: CDF of peers upload capacity

Performance Metrics. We use the *update discard rate*, *update dissemination latency*, and *buffer occupancy* to measure the performance of BCoM. These metrics are defined in our simulation Section 7.4.

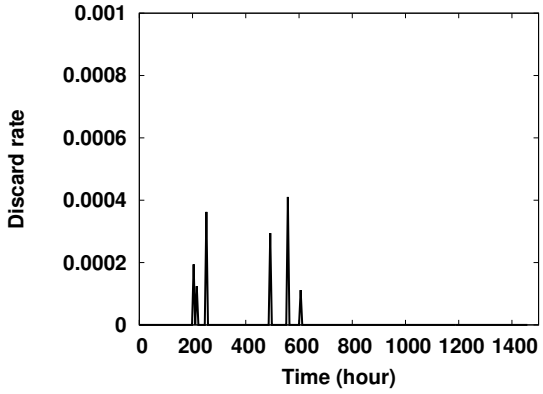

Figure 6.21: The update discard rate

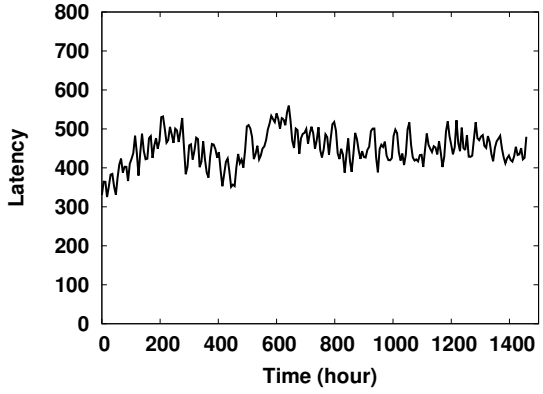

Figure 6.22: The average update dissemination latency (in millisecond)

6.4.3 Performance Results

. Figure 6.21 shows the update discard rate in BCoM. Most of the time BCoM has zero discard rate except two short periods around 200^{th} hour and 600^{th} hour, where BCoM has non-zero discard rates due to update bursts as shown in Figure 6.19. Even though the maximum update discard rate is less than 0.04%, which is negligible in such a large network of more than $6.7 * 10^5$ nodes. Figure 6.22 shows the average update dissemination latency in BCoM, which is kept between 400 and 500 milliseconds. Such fast and stable update delivery confirms the efficiency of the sliding window update protocol with the two enhancements in BCoM. Figure 6.23 shows the average buffer

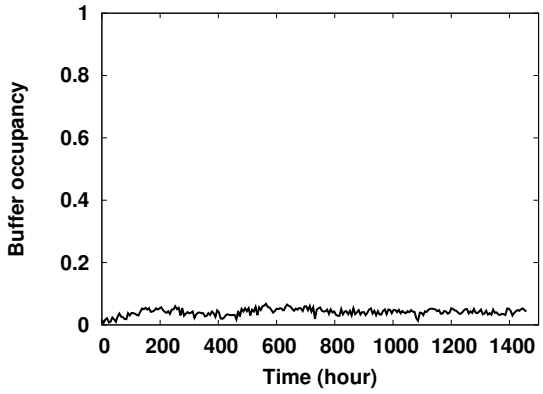

Figure 6.23: The average buffer occupancy

occupancy in BCoM. The buffer size (i.e., window size) is adjusted around 1 to 20 for different objects and dynamic update workloads as shown in figures 6.17, 6.18, 6.19. The buffer size is set for accomodating the slowest node in an update dissemination tree, while most of the nodes have buffer occupancy less than 10% so that the buffer overhead introduced is quite small. All these results demonstrate the applicability and efficiency of BCoM in large-scale social network applications.

Chapter 7

Real-Time P2P Consistency Maintenance

7.1 Real-Time P2P Application Background

In most MMOGs, each player plays a role in a virtual world. Each player is represented in the game by an avatar. A game includes both immutable and mutable objects. Immutable objects (e.g., landscape) are downloaded by game client software. Mutable objects are updated by either players (e.g., avatars, food, tools) or automated algorithms (e.g., NPCs). Every mutable object is represented by its state. For example, an avatar's state may include its position, health, possessions; a tool's state may include its position, shape or content. An NPC's state is similar to an avatar's. The only difference is that each NPC is associated with an automated algorithm for execution. The consistency maintenance updates the game states of all mutable objects correctly for every player. From here on, we use the term "object" to mean a mutable object.

Client-server systems use the primary copy consistency model for maintaining game state. The server maintains a primary copy of every object in the game and

periodically sends updates to players. A game is implemented as a discrete event loop. Each loop iteration is called a "frame." The server broadcasts an update to every player by the end of each frame. The update broadcast includes updates from all mutable objects.

P2P systems follow the primary copy consistency model in client-server systems. Each object has a primary copy where all updates are serialized and sent out. However, all primary copies that were originally maintained by servers are transferred to different players, who are responsible for serializing and delivering updates every frame. Using bandwidth and computation resources from players relieves the reliance on game servers, which thus only do external support, such as authentication. The challenge is how to fully take advantage of the limited resources from a large number of unreliable players to achieve better scalability and robustness compared to a handful of resource abundant and dedicated servers. This challenge motivates the design of PPAct system.

7.2 Rendezvous Enabled Range Query Processing and Subscription

In this section, we first give an overview of how a player obtains its view in PPAct. Then, we present the region partitioning scheme, and the update forwarding scheme. Finally, we present how to map regions to region hosts and objects to object holders.

7.2.1 Overview

We apply AOI filtering to reduce bandwidth consumption due to sending updates. AOI filtering takes advantage of spatial locality of player interests. Thus, each

player gets a confined view in the game, which is an area centered at its avatar's current position with radius R, instead of the entire game map.

The latency bound for acquiring a view is stringent. For example, FPS games require that players' views should be synchronized every 150 ms [35]. This requires a player to receive updates about all objects in its view within every 150 ms. When a player is moving, its view changes accordingly. Hence, the latency bound for receiving an update covers two steps: view look-up and update delivery.

We separate the roles of view discovery from object consistency maintenance by assigning *region hosts* and *object holders*. A region host tracks both objects in a region and players whose views include the region. An object holder maintains the primary copy of an object and sends updates about the object to subscriber players. Players subscribe to all objects within their views for receiving updates. The subscription is performed by sending lookup requests to the region host, because dynamic movements of objects make it difficult for individual players to track all updates. After determining his subscribed regions, a player discovers those region-hosts through contact. In PPAct, region hosts are organized into a two-dimensional DHT overlay based on regions' horizontal and vertical positions. Thus, players are provided with two-hop lookup for frequent continuous movements and logarithmic lookup for the seldom random movements.

To assist region-hosts in tracking objects, object holders register objects with region hosts of their current regions. Once an object moves out of a region's boundary, the holder unregisters from the previous region-host and registers with the new one. No real-time information, such as object's exact location, is involved in the registration. There is no need to update information at the region-host once an object is registered. Therefore, maintenance overhead of registration is low.

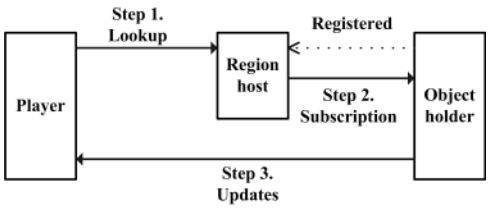

Figure 7.1: The procedure for getting a view

In summary, there are three steps for a player to get a view as shown in Figure 7.1.

1. **Lookup.** A player's view is computed based on its avatar's current position, and a lookup query is sent to the region host of each view region that has not yet been subscribed. An unsubscription message is sent to each region that has gone out of view but previously subscribed.

2. **Subscription.** Every region-host maintains a *subscription list*, recording players who subscribed to this region in the last frame. An *unsubscription list* is maintained similarly. The two lists are sent to each object-holder who has registered with this region.

3. **Update delivery.** After receiving the two lists, each object-holder sends out updates to all subscribers in every frame.

7.2.2 Region Partitioning

The map of a game is partitioned into a set of disjoint regions. A region can be any shape defined by a center point and a radius. We choose circle regions with uniform radius r. Figure 7.2(a) shows an example of region partitioning. The dotted circle represents a player's view where the center is its avatar's position. A player's view consists of seven adjacent regions. In this way, each view region can be flexibly

subscribed when the player is moving and its view is changing.

Region size is an important factor to the efficiency and overhead of subscription, as the subscription is performed in unit of a region. If a region is so large that it covers the entire view, it is hard to accurately define the view. A range query from each player to look up view regions is transformed to a set of exact match queries, each of which locates a view region. Thus, large regions are too rough to approximate the range query. Figure 7.2 (b) shows a region partitioning where three regions cover a player's view. It is possible that some objects are inside the three regions but outside of the player's view. They will be unnecessarily included in the player's range query which wastes network resources. On the other hand, if a region is so small that a view consists of dozens of regions, the excessive overhead of query processing and object registration hand-off will degrade performance.

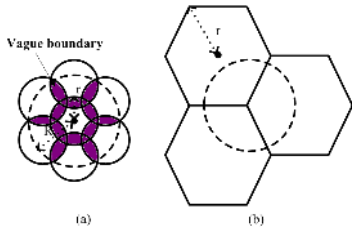

Figure 7.2: Examples of region partitioning

Vague Region Boundary. We apply *vague boundaries* between neighboring regions to reduce hand-off overhead when objects are frequently moving back and forth across boundaries. The shaded area in Figure 7.2 (a) shows the intersections of neighboring regions. The shaded area is called the "vague boundary" since there is no clear borderline between neighboring regions. An object-holder switches registration only when the object enters a new region and moves out of the shaded area. Objects in the shaded area delay switching registrations to avoid unnecessary hand-offs. Such delayed

switchings do not negatively affect the results of range queries. The objects, which are excluded from the query results because of vague boundaries, reside at the border of the player's view. They are less visible than other closer objects that are returned in the results anyway. Hence, the player will not notice the difference.

7.2.3 Update Forwarding and Burst Handling

Limited uplink bandwidth and heterogeneous object popularity give rise to the AOI "hot spot" problem [36, 116], where some object-holders are overloaded by an excessive number of subscribers. Other object-holders with surplus uplink bandwidth may only have a handful of subscribers or perhaps none at all.

In PPAct, we balance workload by making subscribers contribute their spare bandwidth to forwarding updates to other subscribers. The procedure of update forwarding is described as follows:

First, each subscriber attaches a forwarding quota to its lookup query for each of its view region. A subscriber's forwarding quota is equal to its available bandwidth divided by the size of an update, where the available bandwidth is the total bandwidth subtracted by the bandwidth reserved for serving as a region host if it is. The forwarding quota indicates how many recipients a subscriber can forward to under the one hop delay constraint. A frame in FPS is no more than 150 ms and in RPG may be up to 180 ms [30]. Within a frame, at most three steps – subscribing, update delivering, and update forwarding should be completed. Thus, on average each step can use 1/3rd of the time, and one hop delay is set to be 50 ms to satisfy both categories. The measurement statistics of P2P online games in [21] confirm the feasibility of this setting, where more than 80% one hop delay is less than or equal to 50 ms.

Second, after a region host collects the forwarding quota from all subscribers

of the region, it evenly distributes the forwarding quota to every object holder in this region. Thus, the subscription list that a region host sends to each object holder in the region includes all subscribers and their forwarding quota dedicated to the object.

Finally, after obtaining the subscription list, an object holder selects a subset of subscribers as forwarders. It then sends update to each forwarder with a forward list according to each forwarder's quota. The number of forwarders is the maximum number of recipients the object holder can send to under the one hop delay constraint. Subscribers with larger forwarding quota are preferred when selecting forwarders.

Each region host evenly distributes the forwarding quota among object holders because all objects in a region are subscribed as a whole and each object holder needs the same amount of bandwidth for sending updates.

When more peers subscribe to a region, more forwarding quota is contributed to sending updates about the region. Therefore, with the forwarding scheme, the amount of bandwidth contribution is proportional to the demand for sending updates. Moreover, our forwarding scheme avoids sending updates to non-subscribers. We only use one-hop forwarding due to the extended delay and unreliability of multi-hop forwarding.

Temporary update bursts may congest region-hosts or object-holders. PPAct provides a differentiated subscription service to adjust bandwidth consumption when needed. Differentiation is applied to the frequency of receiving updates and the number of objects to be subscribed. Specifically, when a region-host finds that updates cannot be delivered to all subscribers on time, it halves the update delivery by dividing the subscribers into two groups and sends updates to each group every two frames. Such tuning is only applied as a temporary solution and the normal subscription is resumed immediately after bursts.

7.2.4 Mapping Regions to Region Hosts

We select peers (i.e., players) to be candidate region hosts by using the scheme presented in Section 7.3.2. Such candidate region hosts form a DHT, called candidate host DHT. Each candidate host in the DHT has an ID, called candidate ID. Each region also has an ID, called region ID. We use its x and y coordinates as its region ID to preserve the locality of neighboring regions in the 2D-DHT.

When a region (x, y) needs a host, a candidate host is selected from the candidate host DHT by hashing the region ID (x, y) into a candidate ID. The candidate host then becomes the host of the region (x, y) and uses the region ID (x, y) to join the 2D-DHT. If this newly selected host still has spare bandwidth satisfying the threshold requirement defined in Section 7.3.2, it stays in the candidate host DHT. Otherwise, it leaves the candidate host DHT.

A failed region host is replaced by a candidate host instead of using the DHT failure handling in 2D-DHT to prevent a host from maintaining multiple neighboring regions. Hot regions are always clustered together and can easily overload a host if they share the same host. With new hosts selected from the candidate host DHT through hashing, it is less possible for neighboring regions to share the same host.

The maintenance overhead of the candidate host DHT is low because only stable nodes join it, and no extra communication or computation is needed.

7.2.5 Mapping Objects to Object Holders

Each object has an object ID, and each object holder has a holder ID. All object holders form an object holder DHT. Each object has a primary copy, which is maintained by its object holder. Avatars and NPCs are treated differently. The object

Table 7.1: A summary of region hosts and object holders

	Region hosts	Object holders
Major Functions	Track objects and players in a region.	Maintain object consistency and send updates.
Workload Balance	Threshold selection, region assignment.	Subscribers help forward updates.
Failure Handling	Replaced by a new host from the candidate host DHT.	Follow the DHT failure handling.

holder of an avatar is its player since it is mostly accessed by its player. The object holder of an NPC is selected from the object holder DHT by hashing the object ID. When an avatar's holder (i.e., its player) fails, we do nothing because the avatar is gone. When an NPC's holder fails, a new holder is selected according to the DHT failure handling in the object holder DHT.

Maintenance overhead of the object holder DHT is low because only stable peers join and no extra communication or computation is needed. Using the object holder DHT to store object information takes advantage of DHT data replication and failure handling to ensure availability of NPCs.

We use an uptime threshold to select reliable peers as object holders, which form the object holder DHT. We use the update forwarding scheme (as described in Section 7.2.3) to prevent a holder from being overloaded by sending updates about the objects it holds.

7.2.6 Summary of Region Hosts and Object Holders

Table 7.1 summarizes the features of region hosts and object holders.

Figure 7.3 shows the procedure for selecting players to be region hosts or object holders. We first assign each player as the object holder of its avatar. Then, we examine each player using the region host selection scheme presented in Section 7.3. If it has

passed the selection threshold, it becomes a region host. Whether it is a region-host or not, a player becomes an object holder of NPC when it is identified as stable and has spare bandwidth. We use an uptime threshold to identify a stable player. If the uptime of a player is above the threshold, the player is identified to be stable; otherwise, it is unstable. The threshold for object holders is set to rule out the short-live unstable players. If a player leaves the DHT only a short time after joining, failure recovery overhead will outweigh its service. In simulation, we set the threshold for object holders to 10 minutes based on the churn studies in [188, 87].

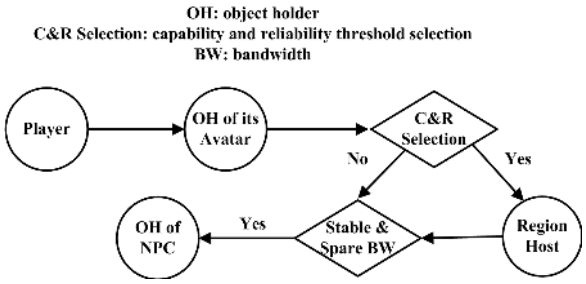

Figure 7.3: The procedure of selecting a region host and an object holder

7.3 Region-host Organization and Selection

In this section, we introduce how to organize regions into a 2-D DHT and how to route lookup queries. We then analytically model the region host selection with both reliability and capability threshold requirements.

7.3.1 Region Host DHT

Regions are organized into a multi-dimensional DHT that we use to process real-time lookup queries, where the number of dimensions in the DHT equals to the number of dimensions in the query attribute. As PPAct works on two-dimensional

geographic queries, we organize regions into a two-dimensional DHT (2D-DHT) with two groups of DHTs, G_x and G_y. G_x is the group of DHTs built on x-axis, and G_y is the group of DHTs built on y-axis. Each DHT in $G_x(/G_y)$ is built on a set of regions that share the same $y(/x)$ coordinates. A region (x_0, y_0) is included in two DHTs, one $DHT_x \in G_x$ with $y = y_0$ and the other $DHT_y \in G_y$ with $x = x_0$. Therefore, each region-host has two routing tables: one is built for x-axis (DHT_x) and the other is for y-axis (DHT_y).

Our 2D-DHT design provides fast lookups for players. This is because a region host has leaf sets of DHT_x and DHT_y including all its neighboring regions along either axis. In addition, the movements of players and objects in a game are mostly continuous. Thus, relying on the last-time contacted region host, a lookup takes 2-hops for moving vertically or horizontally and 3-hops for moving diagonally. The worst-case logarithmic-hop lookup is only performed for rare non-continuous movement.

Graphically, DHT_x is built on a row of regions with the same y coordinate as the owner's region and DHT_y is built on a column of regions similarly. As shown in Figure 7.4, routing from a source region A to a destination region B follows a horizontal route in DHT_x of A to a region C with $y_C = y_A$ and $x_C = x_B$, then a vertical route in DHT_y of C to B.

The routing principles in 2D-DHT are the same as in the original DHT. We choose a representative DHT overlay – Pastry [170] as the routing substrate in PPAct. Pastry is widely used, for example, PAST [171] and SCRIBE [46] are built on top of Pastry. The routing in a 2D-DHT consists of two continuous routes, one from a $DHT_x \in G_x$ and the other from a $DHT_y \in G_y$. An example is shown in Figure 7.5, routing from the source region at $(350, 479)$ to the destination region at $(813, 648)$. Given

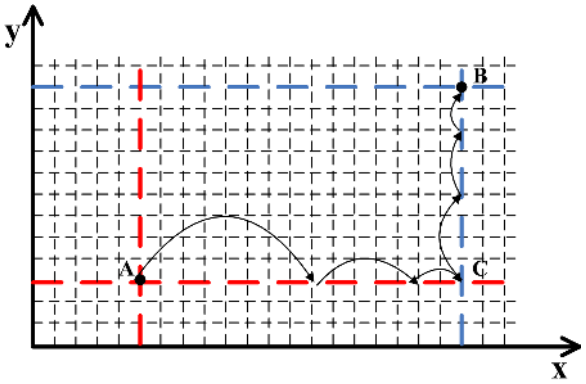

Figure 7.4: Routing from region host A to region host B

a total of $N*M$ regions, a $DHT_x \in G_x$ is built on N region and a $DHT_y \in G_y$ is built on M regions. In the worst case, a route to any region takes $\log(N)+\log(M) = \log(N*M)$ hops. The total number of hops is the same as original DHT routing, independent of the number of dimensions.

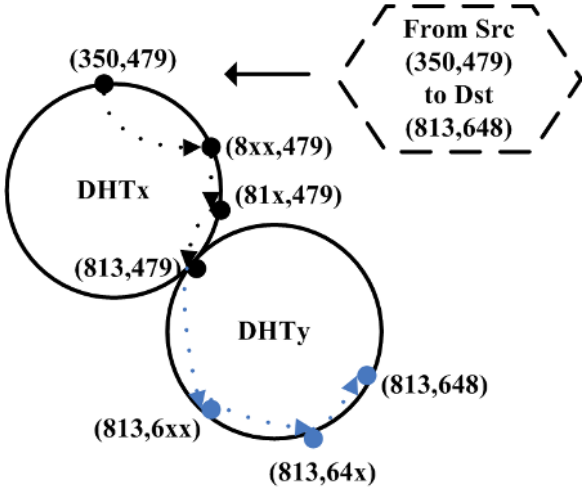

Figure 7.5: An example of PPAct routing

7.3.2 Analysis of Region-host selections

A sufficient level of DHT performance is ensured by maintaining overlay connectivity. This requires that each DHT node should periodically probe other nodes, perform content replication, and execute failure recovery when necessary. Reliable DHT nodes reduce maintenance overhead. If a region host only serves a short time before leaving, failure recovery overhead outweighs its service contribution. In addition, a region host should have enough bandwidth to send subscription lists to object holders on time.

We propose a distributed scheme to select the most stable peers with adequate bandwidth to be region hosts. We focus on selecting peers as candidate region hosts, which form a candidate host DHT (called R-DHT). These candidate region hosts (i.e., R-DHT nodes) are mapped to regions through hashing and join the 2D-DHT as described in Section 7.2.4.

To estimate the number of qualified region hosts, we first analyze the distribution of peers' reliability and capability. Then, we set an appropriate threshold for selection.

We consider a system with n players in steady-state. Each node switches between two states: ON when the node gets online during its uptime and OFF when the node goes offline during its downtime. A node's uptime is the time interval between joining and leaving. Its downtime is the time interval from departure to re-joining. $U(\cdot)$ is the cumulative distribution function (CDF) of the node uptime, and $\bar{u}$ is the average uptime. $D(\cdot)$ is the CDF of the node downtime, and $\bar{d}$ is the average downtime. A node's availability a is measured by the probability that a node is in the ON state, $a = \frac{\bar{u}}{\bar{u}+\bar{d}}$. The expected number of nodes in the ON state is an.

We first examine nodes with expected uptime above T. The information about the node uptime can be obtained either by sampling or by prediction techniques [149, 148]. Let n' be the number of nodes above the threshold T and in their uptime, $n' \leq n$. $\rho = \frac{n'}{an}$ represents the ratio of such nodes, $0 < \rho \leq 1$. $U'(\cdot), D'(\cdot), \bar{u}', \bar{d}'$ and a' represent the counterpart attributes of the nodes with expected uptime above T as $U(\cdot), D(\cdot), \bar{u}, \bar{d}$ and a of all nodes, respectively. Our analysis assumes a node maintains $\log(n')$ overlay connections which is the same as with the general DHT.

Measurement results [175, 130] demonstrate that node uptime is well modeled by a long-tailed distribution. We adopt a shifted Pareto distribution to depict the independence of node uptime as [214]. The probability density function (PDF) of node uptime $u(t)$ is shown in Equation 7.1, where $\alpha > 1, \beta > 0$.

$$u(t) = \frac{\alpha}{\beta}(1 + \frac{t}{\beta})^{-(\alpha+1)} \tag{7.1}$$

And the CDF of the node uptime $U(t)$ is shown in Equation 7.2, where $\alpha > 1, \beta > 0$.

$$U(t) = 1 - (1 + \frac{t}{\beta})^{-\alpha} \tag{7.2}$$

The smaller value of α means a stabler system with longer node uptime. The PDF of selected R-DHT node uptime $u'(t)$ is derived in Equation 7.3.

$$u'(t) = \begin{cases} 0 & t < T \\ \frac{u(t)}{\int_{t=T}^{\infty} u(t)dt} = (1 + \frac{T}{\beta})^{\alpha} u(t) & t \geq T \end{cases} \tag{7.3}$$

And the CDF of the selected R-DHT node uptime $U'(t)$ is derived in Equation 7.4.

$$U'(t) = \begin{cases} 0 & t < T \\ \frac{U(t)-U(T)}{\int_{t=T}^{\infty} u(t)dt} = (1+\frac{T}{\beta})^{\alpha}(U(t)-U(T)) & t \geq T \end{cases} \tag{7.4}$$

The stability of the selected R-DHT nodes is measured by their average uptime $\bar{u'}$ derived in Equation 7.5.

$$\bar{u'} = \int_{t=T}^{\infty} t \cdot u'(t)dt = \frac{\beta + \alpha T}{\alpha - 1} \tag{7.5}$$

The higher the threshold T, the better the system stability $\bar{u'}$. This system stability determines the frequency of maintenance probing and thus the expected overhead.

To calculate the ratio of selected R-DHT nodes over all nodes, we compute the values of n' and ρ as a function of the selection threshold T. According to Little's Law, the average number of nodes in a stable system equals their average arrival rate multiplied by their average uptime in the system. Applying to all nodes in the ON state, we get $an = \lambda\bar{u}$, where λ is the average arrival rates of nodes to the ON state. Applying Little's Law to the selected R-DHT nodes in the ON State, we get $n' = \lambda(1 - U(T))\bar{u'}$. Substituting the variables $an, n', U(T)$ and using $\bar{u} = \int_{t=0}^{\infty} t \cdot u(t)dt = \frac{\beta}{\alpha-1}$, the percentage of selected R-DHT nodes ρ is derived in Equation 7.6, which estimates how many players are qualified for the threshold T.

$$\rho = \frac{n'}{an} = (1+\frac{T}{\beta})^{-\alpha}(1+\frac{\alpha}{\beta}T) \tag{7.6}$$

Next, we examine nodes with bandwidth capacity above a threshold B. The bandwidth consumption of a region-host comes from three aspects: R-DHT maintenance workload B_1, bandwidth reserved as the object-holder of its avatar B_2, and bandwidth reserved as a region-host B_3. $B = B_1 + B_2 + B_3$. PPAct sets probe frequency to be once per average R-DHT node uptime, so that $B_1 = \frac{1}{\bar{u'}} \log(n')p_m$, where p_m is maintenance

packet size, and $\log(n')$ is the number of overlay connections a R-DHT node maintains. B_2 changes with the number of subscribers to the object, and B_3 changes with the number of objects in the region. According to the traffic analysis in Figure 7.14, B_2 and B_3 are empirically set to be 500Kb and 1Mb to satisfy basic demand. Update forwarding and burst handling help deal with the dynamic workloads.

The ratio of nodes qualified for the bandwidth capacity threshold B is denoted as μ. We analyze μ with capacity threshold B in the same way as we analyze ρ with the reliability threshold T. The heterogeneous P2P node capacity is modeled by a bounded Pareto distribution with lower bound L and upper bound H, representing the diverse bandwidth from dial-up modem connections to cable connections. The PDF of the node capacity $f(x)$ is shown in Equation 7.7, where a node capacity is between upper and lower bounds $L \leq x \leq H$, and the shape parameter is γ ($\gamma > 0$).

$$f(x) = \frac{\gamma L^{\gamma} x^{-\gamma-1}}{1 - (\frac{L}{H})^{\gamma}} \tag{7.7}$$

The CDF of the node capacity $F(x)$ is shown in Equation 7.8.

$$F(x) = \frac{1 - L^{\gamma} x^{-\gamma}}{1 - (\frac{L}{H})^{\gamma}} \tag{7.8}$$

Given there are total n nodes with bounded distribution from L to H, the number of nodes ranging from threshold B to H is $(F(H) - F(B))n$. Hence, the percentage μ of nodes selected with capacity threshold B is in Equation 7.9.

$$\mu = \frac{(F(H) - F(B))n}{n} = \frac{(\frac{L}{B})^{\gamma} - (\frac{L}{H})^{\gamma}}{1 - (\frac{L}{H})^{\gamma}} \tag{7.9}$$

Finally, the number of R-DHT nodes qualified for both thresholds is $n \cdot \rho \cdot \mu$. Given the total N regions, PPAct sets the reliability threshold T^* as in Equation 7.10 to

get the best reliability constrained by that the total region-host bandwidth reservation from all selected nodes is sufficient to serve all regions. The average bandwidth capacity of the selected nodes is given by $\bar{b'} = \int_{x=B}^{H} x \cdot f(x)dx$.

$$T^* = arg \ \max T \ \ s.t. \ n\rho\mu(\bar{b'} - B_1 - B_2) \geq N * B_3 \tag{7.10}$$

7.4 Performance Evaluation

7.4.1 Experimental Methodology

We developed a simulator to evaluate the efficiency of PPAct in FPS games and RPGs.

Game Settings. Two game workload generators are developed: one for FPS games and the other for RPGs. Game traffic characteristics are based on trace data of Counter Strike [51] for FPS games and ShenZhou Online games [54] for RPGs, as summarized in Table.7.2. The actions of the players and their movements in FPS games are based on the networked game mobility model [193], which simulates the real FPS player behaviors. Those in RPGs are extracted from the trace ShenZhou Online games in [54], including both player-to-player and player-to-object interactions. The map simulated for FPS games takes an average peer $5 * 10^3$s to walk from one end to the other, and that for RPGs takes an average of $5 * 10^4$s. The map is partitioned into $10 * 10$ regions in FPS games and $100 * 100$ regions in RPGs by default. These default values were selected after careful evaluation of the impact of region size on performance, given in Section 7.4.2.

Network Model. Each player is on an individual machine in a simulated network, representing a general Internet player experience. We adopt the widely used

Table 7.2: A summary of game traces

Trace	Counter Strike	Shenzhou Online
Date	Apr 11 2002	Aug 29 2004
Start Time	08:55	15:00
Period	7 d, 6 h, 1 m	20 h
Established Connections	16030	112369
Total Packets	500 M	1356 M
Mean Payload Size	32 bytes	32 bytes
Mean Packet Size	87 bytes	84 bytes

statistics of the player bandwidth capacity collected at U.S. Broadband report [2]. The upload capacity of game players is shown in Figure 7.6, which is well approximated by a Pareto distribution with a range from 256 Kb/s to 10 Mb/s. We simulate wide geographic areas where players come from. The inter-player round-trip time (RTT) in an n-player game is simulated by drawing n nodes from the Xbox 360 player data set [127] that is spread over the Western United States. The mean, median, and standard deviation of inter-player RTT of this data set are 81 ms, 64 ms, and 63 ms. Vivaldi 3D coordination system [61] is used to extrapolate the RTT values between pairs of players who did not probe each other in the data set. We use a two-state Gilbert model [83], which models packet loss property of Internet paths, setting loss rate to 1% and mean loss burst time to 100 ms.

Performance metric. We mainly use three metrics to measure performance of PPAct in maintaining consistency for real-time games. The *successful action rate* is the ratio of the number of actions completed over the total number of actions issued by players. An action refers to an update that a player issued on an object. An action is completed when the update is successfully received by the object holder on time. The *successful subscription rate* is the ratio of the number of subscriptions received by object holders over the total number of subscriptions requested by players. A subscription refers

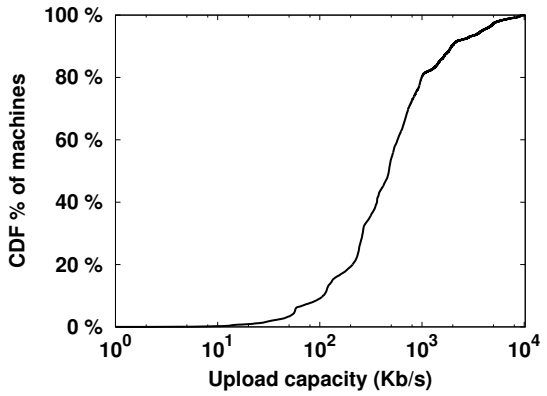

Figure 7.6: CDF of peers' upload capacities

to completion of step 1 and step 2 in Figure 7.1. A player requests a subscription to each object in its view. The *successful update rate* is the ratio of the number of updates received by subscribers over the total number of updates sent out by object holders. An update refers to the completion of step 3 in Figure 7.1. We use 150-ms deadline for FPS games and 180-ms for RPGs according to [30]. Each result shown in the figures is the average of 50 rounds of simulations, and each round executes $5 * 10^3$s.

7.4.2 Evaluation Results

Scalability for FPS Games. We evaluate PPAct for FPS games compared to Donnybrook [35], which is a seminal work on P2P managed FPS games. The successful rates of actions, updates and subscriptions are shown in Figures 7.7, 7.8, 7.9. PPAct outperforms Donnybrook in all three metrics. An important reason is that Donnybrook requires every player to broadcast a guidance message to every other player each second. Getting rid of the broadcast overhead, PPAct saves more bandwidth for delivering updates, actions and subscriptions. Since the broadcast overhead grows exponentially when the number of players increases, PPAct has more advantages over Donnybrook when the system becomes larger.

Comparing results in Figures 7.7, 7.8, 7.9, the successful subscription rate is the lowest of the three. This is because every time a subscription is sent to a different region host, while actions and updates are sent to the same object holders or subscribers until a subscription is changed. Thus, completing a subscription incurs an extra lookup delay over completing an action or an update. The subscription performance is still acceptable because of the constant hop lookup supported by both PPAct and Donnybrook. A Donnybrook subscription takes one hop because the broadcast lets every node know all others. Most PPAct subscriptions take 2 or 3 hops as shown in Figure 7.10. Since players move continuously most of the times, a new region is adjacent to the previously subscribed one. Such a subscription takes 2 hops in 2D-DHT. The 3 or 4 hops are caused by node churn, for another hop is taken to contact the new host. PPAct achieves higher successful subscription rates than Donnybrook under node churn because of overwhelming broadcast overhead in Donnybrook.

To maintain high success rates of actions and updates, we choose a larger subscription area than the player's view. As a result, when a player moves, most of the new view regions intersect with previously subscribed regions. This masks subscription delay.

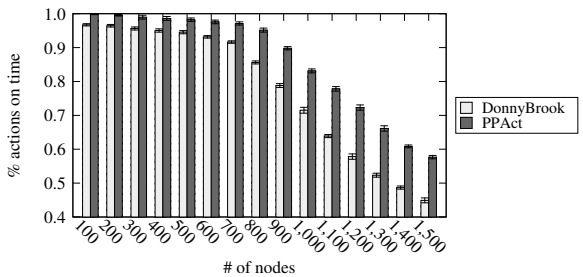

Figure 7.7: Successful action rates in FPS games. Error bars show 95% confidence intervals.

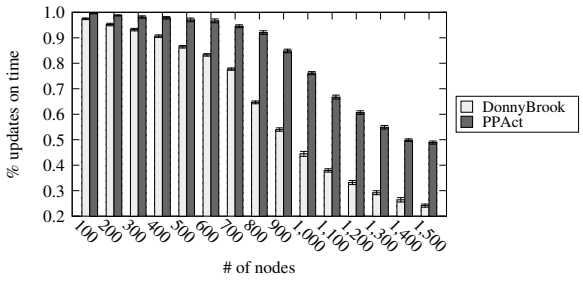

Figure 7.8: Successful update rates in FPS games. Error bars show 95% confidence intervals.

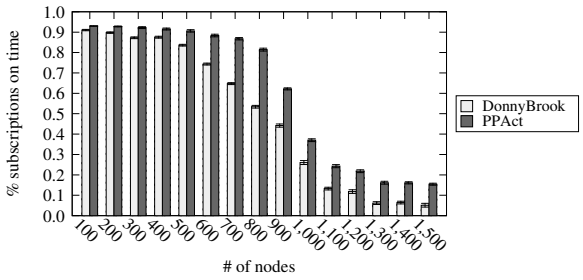

Figure 7.9: Successful subscription rates in FPS games. Error bars show 95% confidence intervals.

Scalability for RPGs. Since Donnybrook does not support NPCs in RPGs, we compare with SimMud [116], a pioneering work on P2P managed RPGs. Both PPAct and SimMud use region partitioning techniques and DHT routing. By default, the number of NPCs is 10^4. Figures 7.11, 7.12, 7.13 show that PPAct achieves significantly better results than SimMud by all three metrics. This is because every SimMud region host must be the object holder for all objects in that region, which may be overloaded easily. PPAct separates the workload of region hosts from object holders to avoid overloading. In addition, PPAct selects reliable and capable players to be region hosts as modeled in Section 7.3.2, while SimMud randomly selects players to be region hosts. As a result, the 2D-DHT in PPAct is more robust and efficient than the DHT in SimMud. As shown in Figure 7.13, SimMud incurs a longer subscription delay than PPAct. This is because SimMud organizes all players into one DHT, and a subscription takes logarithmic hops.

PPAct supports RPGs with both successful action rate and update rate above

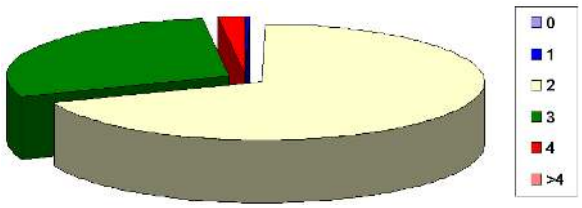

Figure 7.10: Subscription hop counts in PPAct

98% up to 10^4 players. To our knowledge, it is the first one to support P2P managed online games with tens of thousands players.

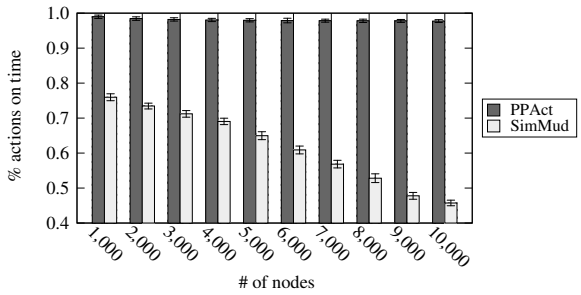

Figure 7.11: Successful action rates in RPGs. Error bars show 95% confidence intervals.

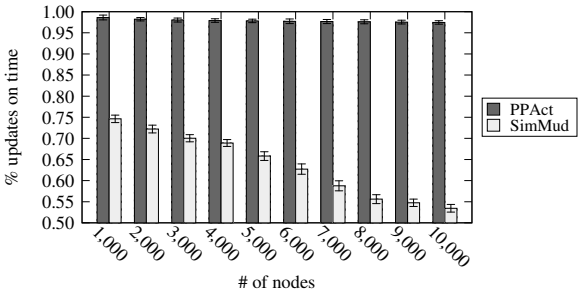

Figure 7.12: Successful update rates in RPGs. Error bars show 95% confidence intervals.

Impacts of population density. Comparing the results in Figure 7.8 and Figure 7.12, the scalability of PPAct is ten times more in RPGs than in FPS games. This is because the density of players in a region is one tenth in RPGs than in FPS games, but the region size is the same in both. The map in RPGs is 100 times larger than that

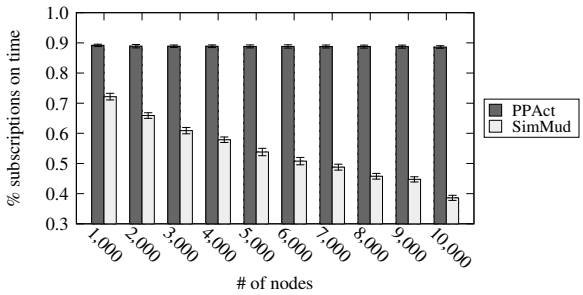

Figure 7.13: Successful subscription rates in RPGs. Error bars show 95% confidence intervals.

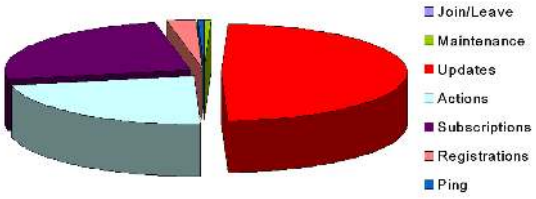

Figure 7.14: PPAct traffic analysis

in FPS games and the number of regions in RPGs is also 100 times more than that of FPS games. When the player population grows 10 times with game map increasing 100 times, the population density becomes 1/10. The reduced population density results in reduced update delivery overhead, which dominates overall traffic in PPAct as shown in Figure 7.14. Increased scalability in PPAct is also a result of our dynamic workload balance scheme. When serving the same 1000 players, the performance of SimMud in RPGs as shown in Figure 7.12 is still lower than that of PPAct in FPS games as shown in Figure 7.8, because SimMud does not handle the AOI "hot spot" problem. Thus, addressing clustered workload of hot regions is critical to scalability.

Impacts of region size. Generally, there are trade-offs in using large regions or small regions. Large regions speed up lookup and reduce query processing overhead, as fewer regions cover the same view and fewer region hosts are queried. However, large regions reduce granularity and accuracy of range query processing. Oversized regions

either waste resources for processing extra areas or provide insufficient views. Oversized regions also increase workload imbalance, since hot areas are covered by a fewer number of regions. To the contrary, small regions lower the workload of each region for better workload balance. A host may choose to take charge of several small regions that are unlikely to have simultaneous workload crowds. Whereas, small regions incur higher lookup overhead and object hand-off overhead (i.e., the overhead incurred by switching registration among regions when an object moves) because objects move across regions more frequently.

We evaluate the impact of region size by partitioning the RPG map into $40*40$, $50*50$, and so on up to $300*300$ regions. The results are shown in Figures 7.15, 7.16, 7.17. Subscription rate in Figure 7.17 improves when the number of regions increases from $40*40$ to $100*100$, and degrades with further increases. The improvement comes from the decreased population density with smaller region size, while further reducing region size imposes excessive subscription overhead when players move. However, the degradation is only reflected in successful subscription rate. The successful action rate and update rate are maintained high as in Figures 7.15, 7.16. Since subscription overhead in PPAct is only a minor part of overall traffic as shown in Figure 7.14, we choose the size of $100*100$ regions as the default region size in our simulations.

Impacts of object number. We evaluate the impact of number of mutable objects by increasing the number of NPCs from 10^4 to 10^5 in RPGs with 10^4 players. The results in Figures 7.18, 7.19, 7.20 show that the performance of PPAct degrades only slightly while increasing the number of NPCs. This is because update delivery overhead is mainly affected by the number of receivers for each update not the number of mutable

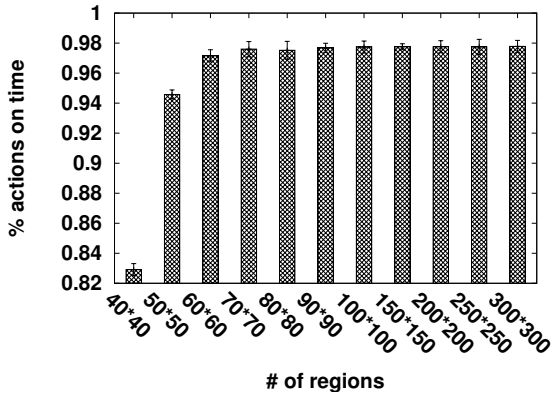

Figure 7.15: Successful action rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

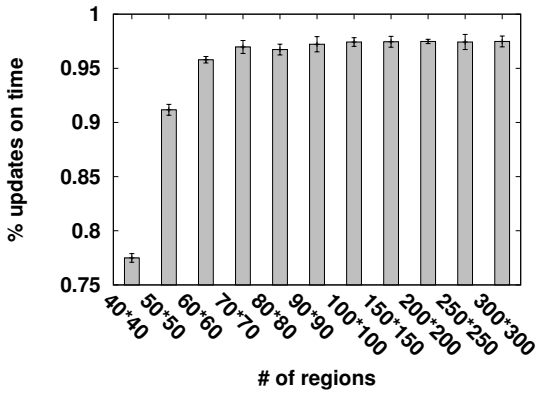

Figure 7.16: Successful update rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

objects. The number of receivers per each update is reflected by the population density. Therefore, even with an increased number of NPCs the overall traffic is in the same order when the population density is kept the same.

Impacts of Churn. According to churn studies in different P2P applications [188], we model the inter arrival time of players by a Weibull distribution with shape parameter $k = 0.6$. We vary the scale parameter to simulate the average leave rate ranging from 10% to 50%. Results under different churn rates do not have presentable differences, so we do not show them separately. All our results are measured under an

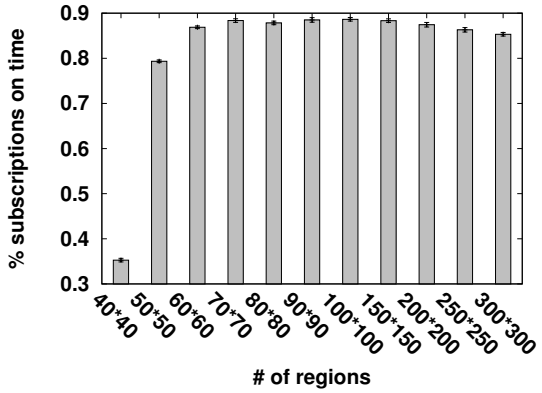

Figure 7.17: Successful subscription rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

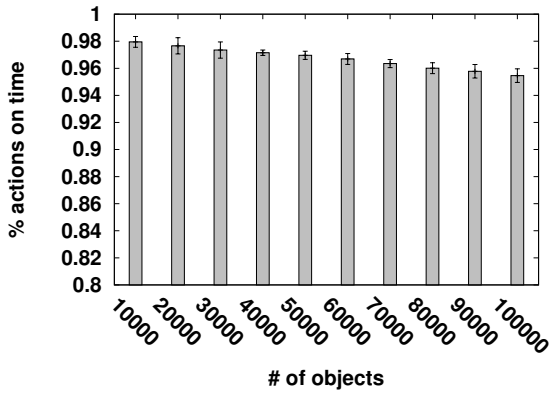

Figure 7.18: Successful action rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

average leave rate of 25%. PPAct performs robustly against node churn because we select reliable players as region hosts. We also observe that short lives of other players do not have noticeable negative effect on overall performance.

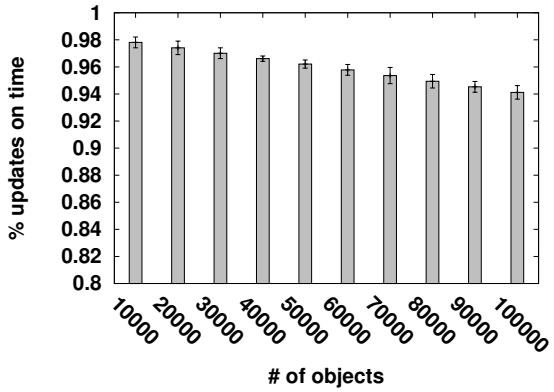

Figure 7.19: Successful update rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

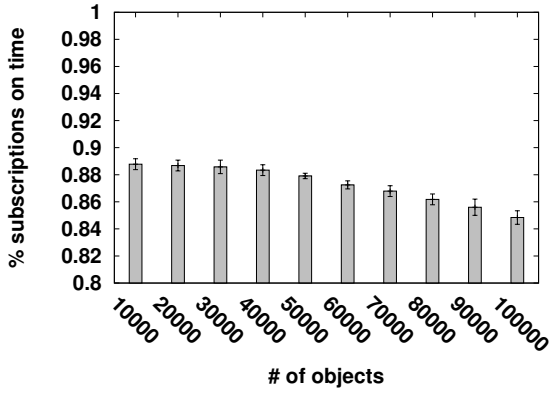

Figure 7.20: Successful subscription rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

Chapter 8

Conclusions

8.1 Contributions

This dissertation makes contributions in the area of P2P support for large-scale user-interacvtive applications. Two key design issues are identified and addressed to improve the scalability and performance of P2P interactive applications. The first issue is how to incentivize users to contribute resources and prohibit free-riders so that a P2P system can function reliably by aggregating resources from a large number of unreliable users. And the second issue is how to provide consistency maintenance for user-interactive applications with different requirements and constraints. In particular, this dissertation makes the following contributions:

I. Satisfying the incentive demands for locating services in P2P systems. Locating service providers for a requested service is one of the fundamental requirement shared by all types of P2P applications. Due to the selfish nature of P2P users and lack of central authorities, peers are not motivated to advertise their services to others or to forward other peers' queries for searching service providers. This dissertation

presented the first time incentive-aware search protocol BuSIS to efficiently locate service providers. BuSIS provides differentiated search service to peers through associating each query with a customized TTL, which is determined by a pre-paid credit budget. In this way, peers who contribute more in helping other peers locating service providers will earn more credits and afford higher budget for their queries to get better search qualities. BuSIS also smoothes search traffic bursts because each peer needs to wait to earn enough credits before issuing a query. Extensive emulations have been conducted at large scale network scenarios to compare performance of BuSIS with flooding and random walk searches with and without selfish user behaviors. The experimental results show that BuSIS always has the lowest search overhead without sacrificing the hit rate. When serving selfish users, flooding and random walk performance degrade dramatically, while BuSIS gracefully keeps the hit rate only with 20% overhead of flooding and 25% of random walk.

II. Meeting the incentive demands for providing services in P2P systems. The selfish nature and lack of central authorities also make peers unwilling to contribute in providing services to other peers. However, P2P systems rely on the coordination of a large number of peers to contribute their services to the P2P community so that their tasks can get completed through the service from the community. This reciprocity principle is the key for P2P systems to function well. This dissertation proposed an indirect reciprocity scheme, called FairTrade, in which peers issue personal currencies to trade services in a P2P system. Personal currency enables indirect reciprocity without relying on any central banks or authorities. It wins extra robustness over global currency as well as much improved trading flexibility and efficiency over direct reciprocity schemes. The acceptance degree of a personal currency depends on the issuer's service capability

and reliance. Peer credit limit is introduced to represent the amount of personal currency that will be accepted by other peers. Every peer as a creditor applies a Bayesian network model to setting peer credit limit for a trading partner peer as a creditee. The Bayesian network model learns the creditee's capability and reliability and anticipates the associated profits and risks for credit setting. Using simulations on a file-sharing P2P system, we demonstrate that FairTrade achieves 100% success rate of download requests without malicious peers, and maintains over 90% success rate even with 50% malicious nodes. The system warms up quickly and does not assume any altruistic service or other kind of help. On average, the system traffic stabilizes before peers issue their second download requests. All these good performances are achieved with extremely low trading overhead, which takes up less than 1% of the total traffic.

III. Addressing the practical issues in providing incentives in P2P systems. Incentive protocols use reciprocity to enforce contribution. Indirect reciprocity schemes are more efficient than direct reciprocity schemes for P2P systems with large population size and high churn rate. However, they lack scalability due to either central banks/brokers or high communication/storage overhead. Besides, they are often vulnerable to malicious attacks. This dissertation proposed an indirect reciprocity scheme, called CoBank, which achieves distributed design, low overhead and strong robustness at the same time. CoBank supports global currency based trading without requiring any special infrastructure to mint currency or manage accounts. CoBank employs a cooperative banking strategy, where the management of user accounts and transactions is done through the cooperation of peers. To ensure account security, each user account is decomposed into several parts stored at different nodes – account holders and each transaction is performed at a third-party peer – transaction arbitrator. Replication

of account data and transaction arbitrator is used to enhance the system robustness. CoBank is scalable because the communication overhead grows logarithmically with the network size and the storage overhead at each node is a constant. It is also resistant to the three types of attacks mentioned above. In addition, CoBank provides incentives for peers to take part in our cooperative banking. The simulation results show that CoBank has an order magnitude lower storage overhead than PledgeRoute [125], as well as much lower communication overhead and latency. Moreover, CoBank maintains more than 90% transaction success rate with 10% of malicious nodes while PlegeRoute has around 65% success rate.

IV. Overcoming the obstacles to maintain consistency for a wide range of P2P applications. A fundamental challenge of efficiently maintain data consistency for P2P systems is to satisfy various application requirements with different resource constraints. This dissertation presented a framework for balanced consistency maintenance (BCoM) in P2P systems with heterogeneous node capabilities and changing workload patterns. Replica nodes of each object are organized into a tree structure for disseminating updates, and a sliding window update protocol is developed for consistency maintenance. An analytical model is presented to optimize the window size according to the dynamic network conditions, workload patterns and resource limits. In this way, BCoM balances the consistency strictness, object availability for updates, and update propagation performance for various application requirements. On top of the dissemination tree, two enhancements are proposed: (1) a fast recovery scheme to strengthen the robustness against node and link failures, and (2) a node migration policy to remove and prevent bottlenecks allowing more efficient update delivery. Simulations are conducted using P2PSim to evaluate BCoM in comparison to SCOPE [55]. The

experimental results demonstrate that BCoM outperforms SCOPE with lower discard rates. BCoM achieves a discard rate as low as 5% in most cases while SCOPE has almost 100% discard rate.

V. Handling the real-time consistency maintenance for P2P applications. An increasing number of P2P applications require real-time consistency maintenance to provide high-quality user-interactive services. A core challenge is propagating updates within stringent time constraints by only using the uplink bandwidth from individual users instead of relying on dedicated servers. This dissertation presented a P2P system called PPAct to provide consistency maintenance for real-time large-scale interactive applications. Massive multi-player online games are used as example applications to illustrate PPAct, but the design can be directly applied to other interactive applications. PPAct adopts the Area-of-Interest (AOI) filtering method, which is proposed in prior works [36, 116] to reduce bandwidth consumption of update delivery. However, PPAct solves AOI's critical problems of bandwidth shortage in hot regions by dynamically balancing the workload of each region in a distributed way. PPAct separates the roles of view discovery from consistency maintenance by assigning players as "region hosts" and "object holders". Region hosts are in charge of tracking objects and players within a particular region, and object holders are in charge of sending updates about a particular object to interested players. Lookup queries for view discovery are processed by region hosts, while consistency maintenance of objects are taken by object holders. This separation distributes the workload and simplifies the lookup procedure and update delivery. Another key idea in the paper is that peers contribute spare bandwidth in a fully distributed way to forwarding updates about objects they are interested in. Thus popular objects for which demands are higher will have more peers forwarding

updates for them. PPAct also presents how to select region hosts and object holders with capability and reliability considerations. A P2P network simulator is developed to evaluate PPAct on two major types of online games: role playing games (RPGs) and first person shooter (FPS) games. The results demonstrate PPAct successfully supports 10000 players in RPGs and 1500 players in FPS games, outperforms SimMud [116] in RPGs and Donnybrook [35] in FPS games by 40% and 30% higher successful update rates respectively.

8.2 Future Directions

Incentive models for multiple co-exist P2P applications. In this dissertation, all incentive models, including search and trading models, are focused on single P2P applications, where the resources or services each user contribute is the same type. To further encourage user participation and contribution, an incentive model for multiple co-exist P2P applications are desired. With such incentive model, there are multiple types of resources a user could contribute in order to get his/her requested services. Since users have more flexibility and can choose their less desirable resources in exchange of more scarce resources, P2P applications can attract more users and have large scalability. The challenge is how to accurately estimate the exchange rate among different types of resources based on changing workload and dynamic user behaviors.

Consistency maintenance for mobile P2P applications. In this dissertation, all consistency maintenance models assume wired network communication. With more and more advanced wireless technology, wireless P2P applications are increasingly popular and important. Wireless P2P applications provide real-time communication in a more convenient way, where wired network may not be available. For example, mobile P2P

evacuation systems can take advantage of users' mobile devices in the emergency area to collect live traffic in that area and identify shortest path in real-time. Such mobile applications impose new challenges in consistency maintenance, such as more limited energy resources and communication interference.

Bibliography

[1] Bit torrent. http://www.bittorrent.com/.

[2] Broadband report. http://www.dslreports.com.

[3] E. weaver. improving running components at twitter. http://blog.evanweaver.com/2009/03/13/qcon-presentation/.

[4] emule peer-to-peer file sharing client. http://www.emule-project.net/.

[5] Face book. http://www.facebook.com/.

[6] Files tube. http://www.filestube.com/.

[7] Fips 180-1, secure hash standard. NIST, US Department of Commerce, Washington D.C., April 1995.

[8] Gnutella. http://www.rfc-gnutella.sourceforge.net/.

[9] Kazaa peer-to-peer file sharing client. http://www.kazaa.com/.

[10] My space. http://www.myspace.com/.

[11] Napster. http://www.napster.com/.

[12] P2PSim. http://pdos.csail.mit.edu/p2psim/.

[13] Ringtonia. http://www.ringtonia.com/.

[14] Roadcasting. http://www.roadcasting.org/.

[15] Scalable network technologies, inc. http://www.scalable-networks.com/.

[16] Title-trader, swap your stuff. http://www.titletrader.com/.

[17] XboxLIVE. http://www.xbox.com/en-US/live.

[18] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999.

[19] E. Adar and B. Huberman. Free riding on gnutella. *FirstMonday*, 5(10), 2000.

[20] N. Adly, M. Nagi, and J. Bacon. A hierarchical asynchronous replication protocol for large scale systems. In *IEEE Workshop on Advances in Parellel and Distributed Systems*, 1993.

[21] S. Agarwal and J. R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *ACM SIGCOMM*, 2009.

[22] P. Albitz and C. Liu. *DNS and BIND, 4th Ed.* O'Reilly Associates, Sebastopol, CA, USA, 2001.

[23] C. Anderson. *The Long Tail: Why the future of business is selling less of more.* New York: Hyperion, 2006.

[24] T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, consistency and practicality: are these mutually exclusive. In *ACM SIGMOD*, 1998.

[25] Martin Angerer, Juergen Huber, Martin Shubik, and Shyam Sunder. An economy with personal currency: theory and experimental evidence. Technical Report 1622, Cowles Foundation, Yale University, 2007.

[26] Panayotis Antoniadis, Costas Courcoubetis, and Robin Mason. Comparing economic incentives in peer-to-peer networks. *COMPUTER NETWORKS*, 46(1):1640–1650, 2004.

[27] C. Aperjis, M. J. Freedman, and R. Johari. Peer-assisted content distribution with prices. In *ACM CoNEXT*, 2008.

[28] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53, 2010.

[29] J. Aspnes and G. Shah. Skip graphs. *ACM Trans. Algorithms*, 3, 2007.

[30] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *NetGames*, 2004.

[31] K. P. Berman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, 1987.

[32] P. A. Bernstein and N. Goodman. The failure and recovery problem for replciated database. In *PODC*, 1983.

[33] D. P. Bertsekas and R. G. Gallager. *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[34] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *ACM SIGCOMM*, 2004.

[35] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *ACM SIGCOMM*, 2008.

[36] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A distributed architecture for online multiplayer games. In *NSDI*, 2006.

[37] Nabhendra Bisnik and Alhussein Abouzeid. Modeling and analysis of random walk search algorithms in P2P networks. In *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, 2005.

[38] Matt Blaze. Key management in an encrypting file system. In *USENIX*, 1994.

[39] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[40] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *Int'l WWW Conf.*, 1995.

[41] L. Breslau, P. Cao, G. Phillips L. Fan, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *IEEE INFOCOM*, 1999.

[42] Eric A. Brewer. Towards robust distributed systems. In *ACM PODC*, 2000.

[43] S. Buchegger, D. Schioberg, L. H. Vu, and A. Datta. Peerson: P2p social networking early experiences and insights. In *EuroSys*, 2009.

[44] Levente Buttyan and Jean-Pierre Hubaux. Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. Technical Report DSC/2001/001, Swiss Federal Institute of Technology - Lausanne, Lausanne, Switzerland, 2001.

[45] P. Cao and C. Liu. Maintaining strong cache consistency in the world wide web. In *IEEE ICDCS*, 1997.

[46] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large scale and decentralized application level multicast infrastructure. *IEEE J-SAC*, 20(8):1489–1499, 2002.

[47] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. Heaphy, and L. A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *IEEE IPDPS*, 2007.

[48] V. Cate. Alex – a global file system. In *USENIX File Systems Workshop*, 1992.

[49] Fabio Celli, F. Marta L. Di Lascio, Matteo Magnani, Barbara Pacelli, and Luca Rossi. Social network data and practices: the case of friendfeed. In *International Conference on Social Computing, Behavioral Modeling and Prediction*, 2010.

[50] R. Chakravorty, S. Agarwal, S. Banerjee, and I. Pratt. Mob: a mobile bazaar for wide-area wireless services. In *ACM MobiCom*, 2005.

[51] Wu chang Feng, Francis Chang, Wu chi Feng, and Jonathan Walpole. A traffic characterization of popular on-line games. *IEEE/ACM TON*, 13(3), 2005.

[52] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, 1996.

[53] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in buliding layered dht applications. In *ACM SIGCOMM*, 2005.

[54] K. T. Chen, P. Huang, and C. L. Lei. Game traffic analysis: an mmorpg perspective. *Computer Networks*, 51(3), 2006.

[55] X. Chen, S. Ren, H. Wang, and X. Zhang. Scope: scalable consistency maintenance in structured P2P systems. In *IEEE INFOCOM*, 2005.

[56] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: a high performance scheme for maintainingreplicated data. *IEEE Transaction on Knowledge and Data Engineering*, 4(6):582–592, 1992.

[57] M. J. Chin, S. Harvey, S. Jha, and P. V. Coveney. Scientific grid computing: the first generation. *Computing in Science and Engineering*, 7, 2005.

[58] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *DIAU*, 2000.

[59] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *ACM SIGCOMM*, 1998.

[60] Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing networks with selfish routing.

[61] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network cooridante system. In *ACM SIGCOMM*, 2004.

[62] F. Dabek, M. F. Kaashoek, Karger D, R. Morris, and I. Stoica. Wide area cooperative storage with CFS. In *USENIX Security Symp.*, 2000.

[63] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *IEEE ICDCS*, 2003.

[64] A. Datta, I. Soica, and M. Franklin. Lagover: latency gradated overlays. In *IEEE ICDCS*, 2007.

[65] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *PODC*, 1997.

[66] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *Int'l WWW Conf.*, 2001.

[67] D. J. Dietterich. Dec data distributor: For data replication and data warehousing. In *ACM SIGMOD*, 1994.

[68] J. Dilley. The effect of consistency on cache response time. *IEEE Transactions on Networking*, 14(3):24–28, 2000.

[69] J. R. Douceur. The sybil attack. In *IEEE IPTPS*, 2002.

[70] Debojyoti Dutta, Ashish Goel, Ramesh Govindan, and Hui Zhang. The design of a distributed rating scheme for peer-to-peer systems. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[71] V. Duvvuri, P. Shenoy, and R. Tewari. Adaptive leases: a strong consistency mechanism for the world wide web. In *IEEE INFOCOM*, 2000.

[72] C. Eikemeier and U. Lechner. Introducing domain specific ad-hoc collaboration: the Peer-to-Peer tool iknow. In *WETICE*, pages 107–112, 2003.

[73] K.L. Calvert E.W. Zegura and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOM*, 1996.

[74] B. Fan, D. M. Chiu, and J. C. Lui. The delicate tradeoffs in bittorrent-like file sharing protocol design. In *IEEE ICNP*, 2006.

[75] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM TON*, 8(3):281–293, 2000.

[76] Daneil Figueiredo, Jonathan Shapiro, and Don Towsley. Incentives to promote availability in peer-to-peer anonymity systems. In *IEEE ICNP*, 2005.

[77] I. Foster and C. Kesselman. *The Grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999.

[78] P. Ganesan, M. Bawa, and H. G. Molina. Online balancing of range-partition data with applications to peer-to-peer systems. In *VLDB*, 2004.

[79] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: Multidimensional queries in P2P systems. In *WebDB*, 2004.

[80] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in dht-based systems. In *IEEE ICNP*, 2004.

[81] Antonio Garcia-Martinez and Michal Feldman. Gnushare: enforcing sharing in gnutella-style peer-to-peer networks. Technical report, Unervisity of California, Berkeley, CA, 2002.

[82] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[83] E. N. Gilbert. Capacity of a burst-noise channel. *The Bell System Technical Journal*, 39, 1960.

[84] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In *ACM PODC*, 2002.

[85] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *IEEE INFOCOM*, 2004.

[86] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *IEEE INFOCOM*, 2005.

[87] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *ACM SIGCOMM*, 2006.

[88] R. A. Golding. *Weak consistency group communication and membership*. PhD thesis, UC Santa Cruz, 1992.

[89] P. Golle, K. L. Brown, and I. Mironov. Incentives for sharing in Peer-to-Peer networks. In *ACM EC*, 2001.

[90] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive replication in peer-to-peer systems. In *IEEE ICDCS*, 2004.

[91] J. Grey, P. Helland, P. O'Neil, and D. Shasha. The dangerous of replciation and a solution. In *ACM SIGMOD*, 1996.

[92] R. Guerraoui, D. Kostic, R. R. Levy, and V. Quema. A high throughput atomic storage algorithm. In *IEEE ICDCS*, 2007.

[93] K.P. Gummadi, R.J. Dunn, S. Saroiu, S. Gribble, H.M. Levy, and J. Zahorjan. Measurement, analysis, and modeling of a peer-to-peer file-sharing workload. In *ACM SOSP*, 2003.

[94] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *ACM SIGCOMM Internet Measurement Workshop (IMW '02)*.

[95] A. Gupta and L. Ying. On algorithms for finding maximum matchings in bipartite graphs. Technical Report 21576(97320), IBM T. J. Watson Research Center, Yorktown Heights, NY, 1999.

[96] R. J. Guy, G. J. Popek, and T. W. Page. Consistency algorithms for optimistic replication. In *IEEE ICNP*, 1993.

[97] M. Ham and G. Agha. Ara: A robust audit to prevent free-riding in P2P networks. In *IEEE P2P*, 2005.

[98] G. Hardin. The tragedy of the commons. *Science*, 162, 1968.

[99] M. P. Herlihy and J. M. Wing. Linearlizability: A correctness condition for concurrent objects. *ACM Trans Programm. Lang. Syst.*, 12(3):463–492, 1990.

[100] R. Hill and R. Dunbar. Social network size in humans. *Human Nature*, 14(1):53–72, 2002.

[101] Y. Hu, L. N. Bhuyan, and M. Feng. Maintaining data consistency in structured P2P systems. *IEEE Transactions on Parallel and Distributed Systems*, 2012.

[102] Y. Hu, L. N. Bhuyan, and M. Feng. P2P consistency support for large scale interactive applications. *Computer Networks*, 56, 2012.

[103] Y. Hu, L. N. Bhuyan, and M. Feng. P2P indirect reciprocity via personal currency. *Journal of Parallel and Distributed Computing*, 72, 2012.

[104] Y. Hu, L. N. Bhuyan, and M. Feng. Cooperative banking assisted P2P incentive design. In submission.

[105] Y. Hu, M. Feng, and L. N. Bhuyan. A balanced consistency maintenance protocol for structured P2P systems. In *IEEE INFOCOM mini conference*, 2010.

[106] Y. Hu, M. Feng, L. N. Bhuyan, and V. Kalogeraki. Budget-based self-optimized incentive search in unstructured P2P networks. In *IEEE INFOCOM*, 2009.

[107] H. Innoue, K. Kanchanasut, and S. Yamaguchi. An adaptive www cache mechanism in the a13 network. In *INET'97*, 1997.

[108] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *ACM PODC*, 2002.

[109] J. R. Jiang, C. T. King, and C. H. Liao. Murex: a mutable replica control schmeme for structured peer-to-peer storage systems. *Advances in Grid and Pervasive Computings*, 3947(3):93–102, 2006.

[110] J. Jung, A. W. Berger, and H. Balakrishnan. Modeling TTL-based internet caches. In *IEEE Infocom*, 2003.

[111] P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K-L Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *ACM SIGMOD*, 2002.

[112] Ar D. Kamvar, M. T. Schlosser, and H. Garcia-molina. Incentives for combatting freeriding on P2P networks. In *EURO-PAR*, 2003.

[113] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *WWW*, 2003.

[114] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *ACM STOC*, 1997.

[115] A. M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *ACM Symposium on Principles of Distributed Computing*, 2001.

[116] B. Knutsson, W. Xu H. Lu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *IEEE INFOCOM*, 2004.

[117] N. Krishnakumar and A. Bernstein. Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM TODC*, 19(4), 1994.

[118] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the world wide web. In *USENIX Symposium on Internet Technology and Systems*, 1997.

[119] B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherence. In *WWW 7 Conference*, 1998.

[120] B. Krishnamurthy and C. E. Wills. Proxy cache coherency and replacement - towards a more complete picutre. In *ICDC99*, 1999.

[121] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, and D. Geels. Oceanstore: an architecture for global-scale persistent storage. In *ACM ASPLOS-IX*, 2000.

[122] A. Kumar. Hierarchical quorum consensus: a new algorithm for managingreplicated data. *IEEE Transaction on Computers*, 40(9):996–1004, 1991.

[123] K.W. Kwong and H.K. Tsang. A congestion-aware search protocol for heterogeneous peer-to-peer networks. *The Journal of Supercomputing*, 36(3):265–282, 2006.

[124] J. Lam, X. Liu, P. Shenoy, and K. Ramamritham. Consistency maintenance in peer-to-peer file sharing networks. In *IEEE WIAPP*, 2003.

[125] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio. A sybilproof indirect reciprocity mechanism for Peer-to-Peer networks. In *IEEE INFOCOM*, 2009.

[126] Karl Reiner Lang and Roumen Vragov. A pricing mechanism for digital content distribution over computer networks. *J. Manage. Inf. Syst.*, 22, 2005.

[127] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye. Measurement and estimation of network qos among peer xbox 360 game players. In *PAM*, 2008.

[128] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *ACM SIGMETRICS*, 2007.

[129] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84, 1979.

[130] D. Leonard, V. Rai, and D. Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. In *IEEE SIGMETRICS*, 2005.

[131] Cuihong Li, Bin Yu, and Katia Sycara. An incentive mechanism for message relaying in unstructured peer-to-peer systems. In *IFAAMAS*, 2007.

[132] D. Li and R. Li. Ensuring content and intention consistency in real-time group editors. In *IEEE ICDCS*, 2004.

[133] M. Li and W.C. Lee. Identifying frequent items in P2P systems. In *IEEE ICDCS*, 2008.

[134] Mingmei Li, Eiji Kamioka, and Shigeki Yamada. Pricing to stimulate node cooperation in wireless ad hoc networks. *IEICE TRANS. COMMUN.*, E90-B(7):1640–1650, 2007.

[135] X. Li and J. Wu. Improve searching by reinforcement learning in unstructured P2Ps. In *ICDCSW*, 2006.

[136] Z. Li, G. Xie, and Z. Li. Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems. *IEEE TPDS*, 19(12):1695–1708, 2008.

[137] N. Liebau, V. Darlagiannis, O. Heckmann, and R. Steinmetz. Asymmetic incentives in Peer-to-Peer systems. In *AMCIS*, 2005.

[138] X. Liu, J. Lan, P. Shenoy, and K. Ramaritham. Consistency maintenance in dynamic peer-to-peer overlay networks. *Computer Networks*, 50(6):859–876, 2006.

[139] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang. Location-aware topology matching in P2P systems. In *IEEE INFOCOM*, 2004.

[140] Z. Liu, H. Hu, Y. Liu, K. W. Ross, Y. Wang, and M. Mobius. P2P trading in social networks: the value of staying connected. In *IEEE INFOCOM*, 2010.

[141] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shanker. Search and replication in unstructured peer-to-peer networks. In *ACM International Conference on Supercomputing*, 2002.

[142] Richard T.B. Ma, Same C.M. Lee, John C.S Lui, and David K.Y. Yau. An incentive mechanism for P2P networks. In *IEEE ICDCS*, 2004.

[143] Richard T.B. Ma, Same C.M. Lee, John C.S Lui, and David K.Y. Yau. Incentive and service differentiation in P2P networks: a game theoretic approach. *IEEE/ACM Transaction on Networking*, 14(5):978–991, 2006.

[144] E. P. Markatos and C. E. Chronaki. A top 10 approach for prefetching the web. In *INET'98*, 1998.

[145] T. F. Martell and R. L. Fitts. A quadratic discriminant analysis of bank credit card user characteristics. *Journal of Economics and Business*, 33, 1981.

[146] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *The 6th annual international conference on Mobile computing and networking* , 2000.

[147] D. S. Menasche, L. Massoulie, and D. Towsley. Reciprocity and barter in Peer-to-Peer system. In *IEEE INFOCOM*, 2010.

[148] J. Mickens and B. Noble. Predicting node availability in peer-to-peer networks. In *SIGMETRICS POSTER*, 2005.

[149] J. Mickens and B. Noble. Exploiting availability prediction in distributed systems. In *NSDI*, 2006.

[150] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollines, , and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57(R.1), HP Laboratories Palo Alto, CA, 2003.

[151] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: a read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[152] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for P2P collaborative editing. In *CSCW*, 2006.

[153] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. In *ACM SIGCOMM*, 1996.

[154] Jaeok Park and Mihaela van der Schaar. Pricing and incentives in peer-to-peer networks. In *IEEE INFOCOM*, 2010.

[155] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *ACM SOSP*, 1997.

[156] M. Piatek, T. Anderson, and A. Krishnamurthy. A case for holistic incentive design. In *Workshop on Future Directions in Distributed Computing*, 2007.

[157] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent? In *NSDI*, 2007.

[158] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe. Contracts: Practical contribution incentives for p2p live streaming. In *NSDI*, 2010.

[159] G. Pierre, M. V. Steen, and A. S. Tanenbaum. Dynamically selecting optimal distribution strategies for web documents. *IEEE Transaction on Computer*, 51(6):1–15, 2002.

[160] D. Pittman and C. G. Dickey. A measurement study of virtual populations in massively multiplayer online games. In *NetGames*, 2007.

[161] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distribute environment. In *ACM SPAA*, 1997.

[162] M. Probst, J. C. Park, R. Abraham, and S. K. Kasera. Socialswarm: Exploiting distance in social networks for collaborative flash file distribution. In *IEEE ICNP*, 2010.

[163] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in internet-like environments. In *ACM SIGCOMM '03*.

[164] M. Rabinovich, N. H. Gehani, and A. Kononov. Efficient update propagation in epidemic replicated databases. In *EDBT*, 1996.

[165] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an internet hosting service. In *IEEE ICDCS*, 1999.

[166] V. Ramasubramanian and E. G. Sirer. Beehive: exploiting power law query distribution for o(1) lookup performance in peer-to-peer overlays. In *NSDI*, 2004.

[167] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.

[168] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the gnutella network: properties of large scale peer-to-peer systems and implications for system design. In *IEEE Internet Computing*, 2002.

[169] T. Roscoe. The planetlab platform. *Peer-to-Peer Systems and Applications*, 3483, 2005.

[170] A. Rowstron and P. Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms*, 2001.

[171] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SOSP*, 2001.

[172] S. Sahi and S. Yao. The noncooperative equilibria of a trading economy with complete markets and consistent prices. *Journal of Mathematical Economics*, 18, 1989.

[173] Y. Saito and M. Shapiro. Optimistic replication. *ACM Computing Surveys*, 5(3):1–44, 2005.

[174] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, 2002.

[175] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *ACM MMCN*, 2002.

[176] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury. Percolation search in power law networks: making unstructred peer-to-peer networks scalable. In *IEEE Computer Socienty P2P*, 2004.

[177] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: a highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[178] M.T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup - hypercubes, ontologies, and efficient search on peer-to-peer networks. In *AP2PC*, 2002.

[179] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *ACM SIGCOMM Internet Measurement Workshop*, 2002.

[180] S. Seuken, D. Charles, M. Chickering, and S. Puri. Market design analysis for a P2P backup system. In *ACM Conference on Electronic Commerce*, pages 97–108, 2010.

[181] H. Shen and C.Z. Xu. Elastic routing table with provable performance for congestion control in dht networks. In *IEEE ICDCS*, 2006.

[182] M. Sirivianos, J.H. Park, X.W. Yang, and S. Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *USENIX*, pages 157–170, 2007.

[183] S. Sorin. Strategic market games with exchange rates. *Journal of Economic Theory*, 69, 1996.

[184] H. Spencer and D. Lawrence. *Managing Usenet*. O'Reilly Associates, Sebastopol, CA, USA, 1998.

[185] K. Sripanidkulchai, B. M. Maggs, and H. Zhang. Effieicent content location using interest-based locality in Peer-to-Peer systems. In *IEEE INFOCOM*, 2003.

[186] H. Stern, M. Eisley, and R. Labiaga. *Managing NFS and NIS, 2nd Ed.* O'Reilly Associates, Sebastopol, CA, USA, 2001.

[187] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM'01*, 2001.

[188] D. Stutzbach and R. Rejaie. Understanding churn in Peer-to-Peer networks. In *ACM IMC*, 2006.

[189] A. J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting behind akami travelocity based detouring. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2006.

[190] Swaminathan Sundaramurthy and Elizabeth M. Belding-Royer. The ad-mix protocol for encouraging participation in mobile ad hoc networks. In *IEEE ICNP*, 2003.

[191] S. Susarla and J. Carter. Flexible consistency for wide area peer replication. In *IEEE ICDCS*, 2005.

[192] K. Tamilmani, V. Pai, and A. Mohr. Swift: A system with incentives for trading. In *P2P Econ*, 2004.

[193] S. A. Tan, W. Lau, and A. Loh. Networked game mobility model for fist-person-shooter games. In *NetGames*, 2005.

[194] X. Tang, J. Xu, and W. C. Lee. Analysis of TTL-based consistency in unstructured peer-to-peer networks. *IEEE TPDS*, 19(12):1683–1694, 2008.

[195] Y. Tang, J. Xu, S. Zhou, and W. C. Lee. m-light: indexing multi-dimensional data over dhts. In *IEEE ICDCS*, 2009.

[196] E. Tanin, A. Harwood, and H. Samet. Using a distributed quadtree index in peer-to-peer networks. *VLDB J.*, 16(2), 2007.

[197] Wesley W. Terpstra, Jussi Kangasharju, Chiristof Leng, and Alejandro P. Buchmann. Bubblestrom: resilient, probabilistic, and exhuastive peer-to-peer search. In *ACM SIGCOMM*, 2007.

[198] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch. Session gaurantees for weakly consistent replicated data. In *ACM PDIS*, 1994.

[199] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *ACM SOSP*, 1995.

[200] S. Tewari and L. Kleinrock. Proportional replication in peer-to-peer networks. In *IEEE INFOCOM*, 2006.

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

[202] Lyn C. Thomas. A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers. *International Journal of Forecasting*, 16, 2000.

[203] A. Thomasian. Two-phase locking performance and its thrashing behavior. *ACM Transactions on Database Systems*, 18(4):578–625, 1993.

[204] Y. Tian, D. Wu, and K. W. Ng. Modeling, analysis and improvement ofr bittorrent-like file sharing networks. In *IEEE INFOCOM*, 2006.

[205] F. J. Torres-Rojas, M. Ahamad, and M. Raynal. Timed consistency for shared distributed objects. In *ACM Symposium on Principles of Distributed Computing*, 1999.

[206] G. Urdaneta, G. Pierre, and M. V. Steen. A decentralized wiki engine for collaborative wikipedia hosting. In *WEBIST*, 2007.

[207] Hal R. Varian. *Microeconomic Analysis*. W. W. Norton Company, New York, U.S., 1992.

[208] M. Varvello, C. Diot, and E. Biersack. P2P second life: experimental validation using Kad. In *IEEE INFOCOM*, 2009.

[209] Vijay V. Vazirani. *Approximation Alogrithm (2nd ed.)*. Springer, Berlin, Germany, 2003.

[210] C. Vecchiola, S. Pandey, and R. Buyya. High-performance cloud computing: a view of scientific applications. In *I-SPAN*, 2009.

[211] V. Vishnumurthy, S. Chandrakumar, and E.G. Sirer. Karma: A secure economic framework for Peer-to-Peer resource sharing. In *P2P Econ*, 2003.

[212] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A robust, tamper-evident, censorship-resistant web-publishing systems. In *USENIX Security Symp.*, 2000.

[213] C. Wang, L. Xiao, Y. Liu, and P. Zheng. Distributed caching and adaptive search in multilayer P2P networks. In *IEEE ICDCS'04*.

[214] C. C. Wang and K. Harfoush. On the stability-scalability tradeoff of dht deployment. In *IEEE INFOCOM*, 2007.

[215] F. Wang, J. Liu, and Y. Xiong. Stable peers: existence, importance, and application in peer-to-peer live video streaming. In *IEEE INFOCOM*, 2008.

[216] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[217] Wei Zhao Wang, Xiang-Yang Li, and Zheng Sun. Design differentaited service multicast with selfish agents. *IEEE Selected Areas in Communications*, 24(5):1061–1073, 2006.

[218] Weihong Wang and Baochun Li. Market-based self-optimization for autonomic service overlay networks. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATION*, 23(12):2320–2332, 2005.

[219] Win Wang and Henning Schulzrinne. Pricing network resources for adaptive applications in a differentiated services network. In *IEEE INFOCOM*, 2001.

[220] Z. Wang, S. K. Das, M. Kumar, and H. Shen. An efficient update propagation algorithm for P2P systems. *Computer Communications*, 30(5):1106–1115, 2007.

[221] Z. Wang, M. Kumar, S.K. Das, and H. Shen. File consistency maintenance through virtual servers in P2P systems. In *IEEE ISCC*, 2006.

[222] H. Weatherspoon, B. G. Chun, C. W. So, and J. Wubiatowicz. Longterm data maintenance in wide-area storage systems: a quantitative approach. Technical Report CSD-05-1404, UC Berkeley, Berkeley, CA, USA, 2005.

[223] K. Wei, Y-F. Chen, A. J. Smith, and B. Vo. Whopay: A scalable and anonymous payment system for Peer-to-Peer environments. In *IEEE ICDCS*, 2006.

[224] M. F. Wong and P. Marbach. Who are your firends? - a simple mechanism that achieves perfect network formation. In *IEEE INFOCOM Mini Conference*, 2011.

[225] C. Wu, B. Li, and S. Zhao. Multi-channel live P2P streaming: refocusing on servers. In *IEEE INFOCOM*, 2008.

[226] B. Yang and H. Garcia-Molina. Ppay: Micropayments for Peer-to-Peer systems. In *ACM CCS*, 2003.

[227] J. H. Yang and Y. L. Chen. A social network based system for supporting interactive collaboration in knowledge sharing over Peer-to-Peer network. *IJHCS*, 66:36–50, 2008.

[228] Q. Yang, G. Thangadurai, and L.N. Bhuyan. Design of an adaptive cache coherence protocol for large scale multiprocessors. *IEEE TPDS*, 3(3):281–293, 1992.

[229] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attack via social networks. In *ACM SIGCOMM*, 2006.

[230] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *OSDI*, 2000.

[231] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. In *ACM SOSP*, 2001.

[232] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. *ACM TOCS*, 24(1):70–113, 2006.

[233] Manaf Zghaibeh and Kostas G. Anagnostakis. On the impact of P2P incentive mechanism on user behavior.

[234] C. Zhang, A. Krishnamurthy, and R. Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *IEEE IPTPS*, 2005.

[235] C. Zhang and Z. Zhang. Trading replication consistency for performance and availability: an adaptive approach. In *IEEE ICDCS*, 2003.

[236] Z. Zhang, S. Chen, and M. Yoon. March: A distributed incentive schemes for Peer-to-Peer networks. In *IEEE INFOCOM*, 2007.

[237] B. Q. Zhao, C. S. Lui, , and D. M. Chiu. Analysis of adaptive incentive protocols for P2P networks. In *IEEE INFOCOM*, 2009.

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

[239] M. Zhao and R. J. Figueiredo. Application-tailored cache consistency for wide-area file systems. In *IEEE ICDCS*, 2006.

[240] C. Zheng, G. Shen, S. Li, , and S. Shenker. Distributed segment tree: Support of range query and cover query over dht. In *Fifth International Workshop on Peer-to-Peer Systems IPTPS*, 2006.

[241] Sheng Zhong and Fan Wu. On designing collusion-resistent routing schemes for non-cooperative wireless ad hoc networks. In *ACM MobiCom*, 2007.