ABSTRACT

MultiKarma: A Fully Decentralized Virtual Multi-Currency

Jon D. Allen, M.S.

Chairperson: Michael J. Donahoo, Ph.D.

Participant-based technologies enable users to contribute resources to a shared pool that in the aggregate provides valuable services, such as social networks, massive multiplayer online games, file exchange, etc. Such systems depend on participant contribution; however, some peers may be unwilling to contribute at a level on par with their consumption. Monetary systems incentivize participation through compensation that allows portability, asynchronous participation, granularity and misbehavior costs. The use of government-backed currencies for incentive structures in participant-based systems results in exchange barriers and high transaction costs, while centralized virtual currencies (e.g., Facebook credits) remove many of the benefits of currency. Karma proposes the use of peer-to-peer systems to create a decentralized, consensus-based currency; however, it lacks a complete specification or implementation. We provide a specification, implementation, and evaluation of Karma. Next, we extend Karma to create a multi-currency system called MultiKarma where participants can mint, manage, and distribute their own currency.

MultiKarma: A Fully Decentralized Virtual Multi-Currency

by

Jon D. Allen, B.A.

A Thesis

Approved by the Department of Computer Science

Donald L. Gaitros, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee

Michael J. Donahoo, Ph.D., Chairperson

Gregory D. Speegle, Ph.D.

Randal L. Vaughn, Ph.D.

Accepted by the Graduate School
August 2011

J. Larry Lyon, Ph.D., Dean

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

ACKNOWLEDGMENTS viii

DEDICATION ix

1 Introduction 1

1.1 Participation Imbalance 2

1.2 Current Solutions 2

1.2.1 Synchronous Solutions 2

1.2.2 Reputation Systems 3

1.2.3 Monetary Systems 4

1.3 Digital Monetary Systems 4

1.3.1 Centralized versus Decentralized Solutions 5

1.3.2 Multiple Currencies 7

1.4 Proposed Solution 7

1.5 Overview 9

2 Related Work 10

2.1 Peer-to-Peer Services 10

2.2 Threats to Peer-to-Peer 13

2.3 Early Solution to Contribution Imbalance 14

2.4 Reputation 15

2.5 Digital Currencies 16
2.6 Karma 19

3 MultiKarma 21
3.1 Original Implementation of Karma 21
3.1.1 Banking Infrastructure 21
3.1.2 Secure Routing 22
3.2 Unaddressed Karma Implementation Issues 23
3.2.1 Update Process 25
3.2.2 Bank Implementation 26
3.2.3 Currency Valuation 29
3.3 Enabling Multicurrency 29
3.3.1 Function Changes and New Attributes 30
3.4 Implementation Details 32
3.4.1 Common API 32
3.4.2 Continuations 33
3.4.3 Transactions in MultiKarma 33
3.4.4 Query 35
3.4.5 Transfer 36
3.4.6 Define the MultiKarma API 38
3.4.7 Dispatchers, Handlers and Messages 39

4 MultiKarma Evaluation 42
4.1 Simulations 42
4.2 Evaluation Metrics 43
4.3 Experiments 44
4.4 Results 44

4.4.1 Transaction Time 45

4.4.2 Message Analysis 48

5 Conclusion 52

5.1 MultiKarma 53

5.2 Future Work 54

BIBLIOGRAPHY 56

LIST OF FIGURES

3.1 An instance diagram of the bank infrastructure. 27

3.2 A visualization of the query transaction protocol. 36

3.3 A visualization of the transfer transaction protocol. 37

3.4 A class diagram for the layered message approach in MultiKarma. 40

3.5 A class diagram of the dispatcher and handler relationship. 41

4.1 A chart analysis of Query execution time. 45

4.2 A chart analysis of Transfer execution time. 47

LIST OF TABLES

4.1 A table displaying min and max executions time for Query 46

4.2 A table displaying min and max executions time for Transfer 48

ACKNOWLEDGMENTS

First I acknowledge God, he has blessed me with the strength and determination to complete this journey.

I would like to express my sincere gratitude to my advisor, Dr. Michael Donahoo for his guidance, knowledge and continued support through this long journey. I express my gratitude to Dr. David Sturgill whose contributions made this possible. I also thank Dr. Greg Speegle who assisted me at the start and graciously accepted being a member of my committee to complete this journey. I would like to acknowledge Dr. Vaughn for participating in my defense committee. I would also like to thank Adam Sealey for his support and knowledge of the thesis process. To my friends and family, I want to thank you for never doubting and always encouraging me that I could complete this journey. Finally, Baylor University for providing me with the opportunity of education, life and love.

DEDICATION

To my wife Barbie, you are always there with a positive word of encouragement. Your endless love, devotion, and support illustrates your pureness of heart. Always and forever.

Love is patient. Love is kind. It does not envy.
It does not boast. It is not proud. It is not rude.
It is not self-seeking. It is not easily angered.
It keeps no record of wrong doing.
It does not delight in evil, but rejoices in the truth.
It always protects, trusts, hopes, perseveres.
Corinthians 13

In honor of my grandfather, Virgil R. Shepard, I can only hope to emulate your drive and sincerity .

CHAPTER ONE
Introduction

Many of the latest Internet technologies have come into existence to share the aggregate resources of their participants, including sharing of bandwidth, storage and computation. Even personal content services, such as contributions to a social network and intellectual contributions to Wikipedia, are a participant-based technology. These technologies commonly share dependence for the resources of their participants to maintain stability, functionality and value. A wide variety of these participant-based technologies have grown, flourished and fundamentally changed user's expectations of Internet services.

One common trait of these participant-based technologies is that the true value of the technology is only realized in the aggregate. For example, social networks, such as Facebook, rely on their participants to post updated information on their profiles and interact with other friends and acquaintances on the site. While a person could maintain an individual website that allows her to post the same type of information, the standalone website fails to provide the same value to its visitors. The social network constitutes a one-stop shop for all new and current information of one's friends.

Because of this dependence on participant participation, many systems utilize incentives/disincentives to ensure participation. We have clearly seen some of these new technologies, such as Facebook, establish natural incentives through the content of its millions of users. In reality though, the vast majority of participant-based technologies struggle daily to draw users and continue to provide value. For example, the once popular MySpace social network site languished after it was unable to create incentives to maintain an active user base.

1.1 Participation Imbalance

Users of participant-based technologies engage in two basic functions: contribution and consumption. Such systems work best when equilibrium between contribution and consumption value is maintained. A wide variation in the value of individual contribution may exist (e.g., expert vs. novice advice). This is not a problem as long as participants understand that is it the value, not volume of their contribution that matters. That is, the efficiency of the system depends on minimizing the gap between consumption and contribution value.

An extreme example of imbalance between contributions and consumption involves *Freeloading*, the consumption of resources without any contribution of resources. Researchers have observed on the popular BitTorrent file sharing network that up to seventy percent of users make no contribution to the network (Adar and Huberman, 2000). Since participant-based technologies are dependent on the contribution of resources, freeloading detracts from the functionality and viability of participant-based technologies.

1.2 Current Solutions

Numerous approaches exist to combat unbalanced contribution. Solutions range from only allowing consumption with simultaneous contribution to depending on the feedback of participants to shun freeloaders. A review of the solutions that follows illustrates that the solutions are incomplete and many are prone to manipulation by ill-meaning participants.

1.2.1 Synchronous Solutions

Some participant-based technologies employ the rudimentary approach of not allowing resource consumption without synchronous contribution (Peng et al., 2008). This approach is typically straightforward to implement but results in curtailing participation in the technology. BitTorrent's *tit-for-tat* is one of the most recognized implementations of such a solution. Other versions of synchronous non-monetary economies, include bartering and gift economies.

Synchronous systems are similar in function to the non-monetary economies and experience many of the same issues as non-monetary economies. William Stanley Jevons explains the issue that occurs in non-monetary economies where two entities must want a resource from each other in order for a transaction to be negotiated. Jevons calls this the *coincidence of wants* (Jevons, 1876). Without an intermediate escrow resource, it can be difficult to execute transactions. While the online world allows for a much larger pool of resources and merchants, it does not completely alleviate coincidence of wants. Synchronous systems have little means to reward a user for the quality or importance of her contribution to the network. In addition, by their very nature, synchronous solutions do not allow for the portability of earned value to another participant-based technology.

Few situations call for equal, synchronous contribution and consumption of resources; consequently, such systems incentivize users to provide the appearance of contribution when they need service. This results in users making low quality contributions just to enable simultaneous consumption. Some systems attempt to address the quality issue by introducing user quality metrics.

1.2.2 Reputation Systems

Another approach to combating this inequality is the use of a reputation system (Gupta et al., 2003). Reputation systems track the quality and reliability of a participant's resources. Users can then utilize this information to gauge the likelihood of receiving a quality resource from a user. Some systems also include level of participation in the calculation of a reputation score. Unfortunately, a reputation metric does not provide a fine-grained scheme for rewarding the wide range of transactions found in participant-based technologies.

Keith Hoffman provides an overview of the possible attacks to reputation systems. While there are ways to mitigate many of the attacks, the short term gains from defrauding reputation can be substantial, and it is difficult to fully remove the incentives to cheat the system (Hoffman et al., 2009). EBay possesses one of the most well-known online reputations systems. Researchers have found that EBay's system has significant value for sellers on the network. Sellers can fetch a ten to fifteen percent premium on their goods compared with lower rated sellers. Unfortunately EBay's reputation system does not illustrate a significant ability to purge unscrupulous buyers from the auction site (Resnick et al., 2003). As a result, a more accountable process to control participant-based misuse is required.

1.2.3 Monetary Systems

Monetary systems allow a user to contribute a resource and then retain credit for the transaction, allowing for asynchronous contribution and consumption. Asynchronous tracking encourages users to provide resources even if they are not currently consuming (Tamilmani et al., 2004). Users are more likely to contribute resources if they know they are storing credit for contribution. This solves the coincidence of wants problem and allows a more robust economy to emerge.

While some solutions have been proposed, currently deployed monetary solutions are unable to address the full scope of the inequality issue, as they are limited in portability, granularity, and encumbered with transaction fees. Today's asynchronous solutions depend on government-backed currencies resulting in transaction fees (e.g., PayPal) or are dedicated to a small scope of participant based technologies limiting their portability (e.g., Facebook, Zynga).

1.3 Digital Monetary Systems

Due to the unique characteristics of participant-based systems, traditional monetary solutions fail to meet the requirements of an digital monetary system. In analyzing potential digital monetary solutions, several attributes must be present. First, portability allows for currency to seamlessly transition between participant-based technologies. Second, transactional costs for currency usage must be minimized and ideally eliminated to allow for affordable usage. Third, the solution must allow for currency exchange without incurring transaction costs. Additionally, an ideal monetary system supports *micropayments*, which involve exchanging currency in amounts less than the smallest denomination available in many government-backed currency systems. For example, fair compensation for forwarding a message on a peer-to-peer network might be a fraction of a penny.

One commonly proposed monetary system for online use leverages existing government-based currency systems. Paypal provides the most well known of these solutions and enables storing government-back currency in an e-wallet. While Paypal experiences wide adoption, most e-wallet or systems that leverage government-backed currencies struggle to succeed. First, users tend to sense more financial risk when exchanging a government currency, resulting in decreased likelihood of user participation. Second, many transactions on participant-based technologies fall in the category of micropayments. Finally, transaction costs associated with the exchange of government-backed currencies only compounds the difficulties. A majority of online transactions leverage third-party credit networks that charge both a fixed transaction fee and a percentage of the transaction value. The transaction fee may be several orders of magnitude more expensive than the micropayment value, making such a system unusable.

1.3.1 Centralized versus Decentralized Solutions

Another solution employs a fully centralized digital monetary system in which users rely on a single entity for minting of digital currency. Due to the positive attributes for currency owners, online games, such as Blizzard's World of Warcraft and social gaming publishers, like Zynga, leverage centralized digital monetary systems. Generally, centralized solutions allow for currency to be earned by contribution of a resource, such as time or government-backed currency. One could deduce with 1000 Warcraft gold coins selling for almost ten U.S. dollars that there are millions of dollars in virtual wealth in centralized systems. The main drawback of this approach is that it only allows users to spend currency in the game in which it was earned or redeemed. Though Zynga and Blizzard produce numerous games with similar economic frameworks, portability of currency between games does not exist.

Centralized solutions struggle to meet the desired attributes for participant-based technologies. A single entity controls the valuation of the currency and sets the rules for creation, trading and redemption. While this centralization is ideal for a game publisher or social network owner, it leaves users with a set of rules favorable to the currency owner and a lack of cohesion in digital currencies. Many of the centralized entities even forbid in their acceptable use policies the trading or selling of the virtual currency, thus eliminating the ability to have even rudimentary portability of value. Furthermore, these solutions fail to account for the natural economic factors of inflation and deflation of currency. In addition, in the event, a centralized entity's business fails users generally lose all value stored in the centralized system. To mitigate these issues we look to a decentralized approach.

A completely decentralized monetary system eliminates the concentration of control and enables users to leverage free markets, portability and limitation of transactional costs. These solutions also protect against common online attacks such as a denial of service. In addition, the decentralized approach makes it difficult for an entity (e.g., such as a government) to force the shutdown of the currency system. Later in this chapter, we discuss Karma, a proposal for a fully decentralized digital currency. Decentralized solutions do not need to abandon an entity's ability to have a separate currency.

1.3.2 Multiple Currencies

Today's popular monetary solutions for participant-based technologies limit themselves to a single currency approach. As with real world economies, there are several arguments for providing a multicurrency economy. Friedrich Hayek provides the arguments for currency competition in his work entitled"Denationalisation of Money" (Hayek, 1978). Hayek challenges the notion that there could be too much competition in the issuance of currencies by applying traditional economic theory to currency and showing that competition yields the best outcome for consumers. His argument hinges on the fact that governments profit by centralized control of currency through allocative inefficiency, technical inefficiency, and positive economic profits of monopolistic production of money (Vanhoose, 2011).

For virtual currencies, multiple currencies foster the ability to transition value among participant-based technologies through currency exchange without having to endure significant transactional cost. Present methods of moving value between digital currencies require multiple transactions that include a government-backed currency. This constraint usually forces users to burden a relatively high transaction cost for exchange.

1.4 Proposed Solution

In 2003 Vivek Vishnumurthy published the concept of Karma (Vishnumurthy et al., 2003), a secure decentralized economic framework for currency exchange. In Karma, a distributed set of peers (called the bank set) maintains each user's current balance. Users (and their associated bank set) are identified by a user ID. Given these IDs, users can perform two basic operations in Karma: query and transfer. A query allows a user to discover the current balance of another user's account. A transfer enables two users to move currency between accounts. Karma uses user ID generation based on a cryptographic key pair designed to prevent sybil attacks and enable persistent user identity. Karma's distributed P2P approach protects from traditional denial of services attacks, byzantine attacks, forgery and other attacks that would put the integrity and reliability of the system at risk.

Vishnumurthy's proposal includes some profound concepts but is highly theoretical and fails to provide many of the details and policies necessary to produce a functional implementation (Vishnumurthy et al., 2003). Karma's specification omits significant policy and implementation details required to successfully develop the solution. We provide solutions for Karma's omissions, such as a bank mapping algorithm, specification of signed balances and design for bank instances that prevent transaction collisions. We then apply these solutions to a fully functional implementation of Karma in Java.

In building on Karma, we propose the creation of MultiKarma, a multicurrency-based system that allows users to create currencies, control currency circulation volume, and/or purchase/sell resources on any participant-based technology. Such a system can be used to reward contributors by providing them with currency to trade for future consumption while limiting the ability of strict consumers (i.e., unbalanced participation) through currency deprivation. A new concept that MultiKarma includes is the ability of any user of the network to create and manage a currency. This attribute encourages existing and emerging participant-based technologies to set themselves apart economically while still sharing a common economic framework.

A currency owner can track the key statistics for her currency and influence the market to balance inflation and deflation. For example, a company like Disney can create a currency, Disney dollars. Disney is then able to reward participation for locally caching content on a user's computer for use by nearly consumers. A participant may then use those Disney dollars to purchase Disney content or exchange the dollars for another currency on the network. Disney controls valuation of its currency through supply and exchange.

One deterrent of many economic frameworks is their inability to provide portability across all participant-based technologies. MultiKarma works to overcome this through multiple currencies that enable currency exchange. While a single currency solution is simpler for portability, it prevents other attributes we want to provide. By having multiple currencies, MultiKarma allows currency minters to maintain control of their currency. This allows for applications utilizing the framework to service a wide array of participant-based technologies. MultiKarma natively allows exchange of currencies between users without incurring transactional costs.

1.5 Overview

The chapters that follow address our implementation of original Karma as well as the review, implementation and evaluation of MultiKarma. Chapter two reviews the existing contributions to digital currency frameworks, reputation and distributed banking infrastructure. Chapter three discusses the implementation decisions and specific requirements that enable the creation of MultiKarma. Chapter four evaluates MultiKarma through simulation to evaluate system performance. Chapter five presents conclusions based on our work and proposes future work in distributed currency.

CHAPTER TWO

Related Work

The use of peer-to-peer systems has grown from simple file sharing (e.g., Napster) to critical communication infrastructure (e.g., Skype). Unlike the traditional asymmetric client/server model, peer-to-peer systems assume that end systems contribute services, where optimally the levels of contribution and consumption are equal. Consequently, one of the major threats to the success of a peer-to-peer service involves nodes that consume much more than they contribute. Existing research and systems propose a wide range of solutions.

2.1 Peer-to-Peer Services

Peer-to-peer technology was the talk of only geeks and hobbyists until the founding of Napster (Giesler et al., 2003) by Shawn Fanning in June of 1999. Napster's music sharing service provided the first glimpse into the power of peer-to-peer. Though later shuttered for legal reasons, Napster opened the floodgates for the adoption of peer-to-peer technologies. In 2001 Kazaa (Liang, 2004) followed the path blazed by Napster to introduce a fully decentralized peer-to-peer file sharing service. The service expanded file sharing to any file type and was not easily closed due to the robustness provided by its decentralized structure. During the same year, Freenet (Clarke et al., 2001) was first specified providing one of the first services to illustrate the expanded possibilities of peer-to-peer networks. Freenet not only protected the anonymity of producers and consumers but also allowed users to contribute resources without an awareness of how the resources would be used. This characteristic brought significant dialog around the implication of participating in illicit activity. More importantly it highlighted the ability of peer-to-peer technologies to stretch the boundaries of possibilities in networking technology.

As time passed, additional peer-to-peer technologies were defined that added new classes of service to the landscape. Secure Overlay Services (SOS) (Keromytis et al., 2002) was developed with a primary goal of creating a reliable distributed network that applications can leverage for critical communication, such as emergency notification. The solution leveraged a peer-to-peer infrastructure to create an abstraction from endpoint locations to prevent denial of service attacks. Such abstraction allowed for the communication between nodes using the overlay layer rather than through more direct Internet Protocol addressing. Internet Indirection Infrastructure (Stoica et al., 2002) leveraged the concept of an overlay network to provide networks services traditionally not supported by the lower layer network. Solutions like these illustrated the ability to layer new robust services on top of simple peer-to-peer technologies. Regardless of underlying network support for the features, overlay solutions were able to support more advanced networking protocols such as multicast and concast. As a result, newer peer-to-peer applications often to create virtual network stacks on top of the traditional TCP/IP stack.

One fundamental peer-to-peer advancement that enabled overlay storage and abstracted routing networks was the Distributed Hash Table (DHT). DHT nodes are identified by fixed-length hash values, rather than by IP addresses. A properly implemented DHT enables the ability to support secure overlay routing, which allows anonymous, secure message transfer. While a node can identify the destination address for the purpose of routing, the true network identity (i.e., IP address) is not revealed between endpoints. These characteristics allow for a new generation of peer-to-peer applications. Content Addressable Network (CAN) highlighted these features with the statement, "The Distributed Hash Table (DHT) functionality supported by CAN serves as a useful substrate for a range of large distributed systems; for example, Internet-scale facilities such as global file systems, application-layer multicast, event notification, and chat services can all be layered over a DHT system" (Ratnasamy et al., 2001).

A similar, but more widely adopted DHT, is Pastry (Rowstron and Druschel, 2001) developed by researchers at Microsoft Research and Rice University. The team went on to expand the project by layering an abstracted file storage system on top of Pastry known as Past (Druschel and Rowstron, 2001). An additional service, Scribe (Castro et al., 2002), runs on Pastry to provide multicast communication capabilities. Pastry's goal is to provide application-layer routing and to enable advanced peer-to-peer applications. Each node in the Pastry overlay connects to a group of nodes known as its *neighbor-set*. Each node is assigned a unique 128-bit fixed-length binary value referred to as a *nodeID*. Pastry's overlay is dynamic and adjusts for nodes both joining and leaving the network. Routing is accomplished by forwarding packets to a node whose nodeID is numerically closer to the destination ID. In addition, Pastry accounts for geographic distance and attempts to minimize routing hops. Pastry and other peer-to-peer overlays provide robust function, but the untrusted nature of their participants leaves them susceptible to numerous attacks.

Miguel Castro explores the security risks for structured overlay peer-to-peer networks in "Secure Routing for Structured Peer-To-Peer Overlay Networks" (Castro et al., 2002). The work concludes that the development of a secure routing overlay that abstracts network identifiers from application communication mitigates many of the risks the networks face. The proposal goes further to assert that a proper secure routing overlay can guarantee message delivery even with a percentage of malicious nodes. In the end, truly achieving the concept of a secure routing overlay has proven difficult. While version 2.1 of Pastry has been unable to provide fully secure routing, it is one of the stated goals for an upcoming release. Due to Java support, wide adoption and support for the *Common API*, we will use Pastry to implement our proposed solution.

One of the early challenges for developers writing applications for overlay networks was the inability to easily move applications from one network to another. In 2003 researchers from MIT, Rice and UC Berkley proposed the Common API (Dabek et al., 2003) as a solution to enabling application to overlay transparency. The Common API outlines a set of functions needed to provide key-based routing. Each overlay network then provides an implementation of the Common API. This approach limits the ability to implement functionalities unique to a particular DHT or overlay network. In most cases, the benefit of seamlessly transitioning overlay networks outweighs such limitations. As the Common API continues to expand and provide more advanced functions, the disadvantages are also likely to decrease. Overlay networks and the Common API simplify implementation of peer-to-peer applications but do not appear to improve their ability to defend against the threats that challenge them.

2.2 Threats to Peer-to-Peer

Peer-to-peer networks need the contribution of resources to function and thrive. Unfortunately multiple threats could impede these networks functionality. As discussed previously, Freeloading is the first and most concerning threat. One study "Free Riding on Gnuetella" (Adar and Huberman, 2000) shows that seventy percent of users share no files at all. An additional study found the top one percent of sharers generates fifty percent of all search responses (Saroiu et al., 2002). On a related note, such a limited and concentrated set of contributors illustrates another vulnerability of many peer-to-peer networks to a successful denial of service attack on only a small percentage of users.

The popularity of peer-to-peer networks for copyright infringement resulted in new ideas that threaten to disrupt the networks. One of these is the pollution of resources, a relatively simple way to disrupt the reliability and quality of peer-to-peer networks. Malicious users publish resources that appear to be legitimate. Only after fully consuming the resource does the user realize the resource is unusable. The most surprising attribute of the network pollution attack is that only a small percentage of files need to be polluted to significantly impact the perception of the content available on the network (Christin et al., 2005).

2.3 Early Solution to Contribution Imbalance

Gnutella provides one of the first networks to address the freeloading issue by implementing a strategy that tracks contribution and consumption. Exchange of resources must occur at a similar rate or consumption is throttled. This solution is known as the *Tit-for-Tat* strategy (Peng et al., 2008). Several deficiencies exist with this solution. First Tit-for-Tat is limited to only synchronous service exchange. It does not allow for users to store the value of their contributions and, instead, requires that redemption of contribution occur immediately. Second it does not address the issue of network pollution. Even worse, Tit-for-Tat may actually reward users uploading polluted resources.

Another approach is to account for resource usage through the use of escrow services where users exchange services through an intermediary. The intermediary, as well as the owner of the resource, is compensated during the exchange process. Escrow services as proposed in “Escrow Services and Incentives in Peer-to-Peer Networks” (Horne et al., 2001) are primarily geared to file sharing content. This escrow concept could be extended to mitigate network pollution by holding compensation for contribution until the consumer is satisfied with the quality of the resource.

As we review the contribution imbalance landscape, solutions vary from metrics of trust to economic frameworks. Many of the early solutions to peer-to-peer threats can be classified as reputation systems. Regardless of where solutions land on the digital currency spectrum, there is clearly an implicit need for reputation in participant-based technologies.

2.4 Reputation

Reputation is most simply defined as measuring the reliability and trust of an entity. The Internet, online commerce and participant-based technologies cultivate a consumer desire to have reputation metrics. There are two fundamental approaches to calculating the online reputation of an entity. The first is an automatic system, which uses an algorithmic policy to calculate trust. The second is a manual type system, which allows the feedback of participants to determine an entity's reputation. Manual systems are generally preferred in high risk and/or financial transactions where the trust and reliability of an entity is not a simple calculation.

Online commerce sites, such as EBay, leverage the attributes of a manual reputation system. Automatic systems address situations where the relative cost of user contribution is high. Routing of packets on a peer-to-peer network presents] an ideal case for an automatic reputation system. Reputation systems have value and as such are prone to a number of attacks to inflate or devalue an entity's reputation. Collusion is one type of attack in which fake transactions can be used to falsely inflate the reputation of an entity. A second attack is to use false feedback to either positively or negatively affect an entity's reputation. A proposed solution to this attack utilizes Eigentrust (Kamvar et al., 2003). Eigentrust attempts to use a reputation algorithm to filter out nodes submitting false content to peer-to-peer networks. Like Eigentrust there are a number of niche solutions that address the need for reputation on particular participant-based technologies while mitigating reputation attacks. XRep (Damiani et al., 2002) and DCRS/CORC (Gupta et al., 2003)are two such niche solutions that address freeloading on peer-to-peer networks. Some researchers propose more generic reputation frameworks. Two such solutions are PeerTrust (Xiong and Liu, 2004) and TrustGuard (Srivatsa et al., 2005), which provide excellent solutions for reputation when paired with a generic currency framework.

Thus far, the solutions reviewed to combat the threats facing peer-to-peer networks are difficult to apply to all participant technologies. A currency solution provides compensation for the contribution of a resource and certainly lays the foundation for an asynchronous incentive/disincentive system.

2.5 Digital Currencies

Soon after the broad deployment of the Internet to everyday users, a need for online currency arose. Several commercial solutions attempt to address this need in the early days of the Internet. Flooz (Flooz, 1999) was one of the first digital currency systems and is best described as generic gift certificates. Flooz certificates could be purchased at the Flooz.com website and then many e-tailers would accept the digital currency for purchases. The company made a hard push during the 1999 holiday season using Whoopi Goldberg as a spokesperson, but filed for Chapter 11 bankruptcy soon after. Mounting consumer complaints and financial troubles brought the company to its demise in the summer of 2001. Beenz was another of the initial digital currency systems. It allowed participants to earn "Beenz" (Beenz, 1999) for activities as a type of reward. The rewards could be gained for as little participation as visiting a website or making a purchase at an online store. The "Beenz" could then be redeemed at participating online e-tailers and at the Beenz online store. Beenz also was forced to close as a result of financial issues only one week after its competitor Flooz.

Another online currency approach is to store government-backed currency in an *e-wallet*. The wallets typically provide some level of fraud protection and also limit financial risk by only exposing the funds deposited for expenditure. This results in a decrease in users perceived risk during online transactions since they are using a protected wallet. PayPal is the most widely adopted e-wallet solution. It allows users to hold government-backed currency in an account or withdraw it from a number of sources for Internet-based transactions. MonCash (Moncash, 2011) is another online e-wallet solution. The system allows users to populate their wallet and spend the currency at sites that accept MonCash. These solutions do present several challenges and shortfalls. E-wallet solutions charge transaction fees. This characteristic tends to limit the appeal of this type of solution for micropayments. In addition, the online services and resources that can be exchanged are rarely valued in levels that equate with government-backed currency denominations. Ideally any online currency solution should be able to address traditional government-backed currency needs as well as address the new challenges presented by the participant-base technologies such as micropayments. Ultimately both electronic currency and e-wallets are for convenience and risk minimization.

Over the last few years, there has been a resurgence of activity in the area of digital currency. The exchange of government-backed currency for digital currency and goods has become commonplace. This demand for digital currency has created an emerging industry called *Gold Farming*. "From the start of the 21st century, a new form of employment has emerged in developing countries. It employs hundreds of thousands of people and earns hundreds of millions of dollars annually. Yet it has been almost invisible to both the academic and development communities. It is the phenomenon of "gold farming": the production of virtual goods and services for players of online games" (Heeks, 2008).

The popularity of massive multiplayer online games (MMOG) and social networks has accelerated the demand for virtual goods and currency. Some publishers of games and online content have embraced the idea of exchanging government-backed currency for digital currency. In 2005, Sony Online Entertainment opened a virtual goods marketplace that allows gamers to purchase virtual goods using government-backed currency (Entertainment, 2011). In late 2009, Zynga Games, a publisher of a number of popular online games for use on the Facebook social network, partnered with Fast Card to produce game cards that could be purchased for virtual in game currency (Zynga, 2011). The common theme across these solutions is that they are for one or a small group of participant-based technologies. Publishers look at the digital currency and goods marketplace as a revenue stream but are unable to provide the unity needed to provide a ubiquitous solution. In late 2008, a Facebook application start-up Jambool attempted to fill the digital currency void. They launched *Social Gold*, a social gaming digital currency platform developed by former Amazon.com employees (Jambool, 2010). Social Gold provides APIs for common web programming languages to enable the management and purchase of currency. This solution was developed to eliminate the need for unique solutions and allow for cross-application currencies. It was also designed to allow micropayments without leveraging crippling transaction fees. The greatest downside for the Social Gold platform is that it is squarely designed for one segment of participant-based technologies, social networks. The platform is also highly centrally controlled and managed giving significant power and leverage to Jambool. Similar to many of its predecessors, Jambool ceased operation of Social Gold in May 2011.

System With Incentives For Trading (Tamilmani et al., 2004), (SWIFT) builds on the Tit-for-Tat system implemented in Gnutella. SWIFT provides a digital currency network that allows consumers and contributors to negotiate for segments of files. The authors illustrate that individuals who contribute significantly are able to consume at a higher rate. While asynchronous, SWIFT thus far only applies on trading file segments and not other functions of peer-to-peer networks. With some adaptation, the concept is extensible to curb network pollution but did not as originally implemented.

The latest iteration of digital currency garnering significant public interested as well as concern is BitCoin (Nakamoto, 2009). BitCoin is a cryptographic digital currency that is created through the contribution of computation cycles. This digital currency gives greater returns to early adopters as the rate of return decreases as more Bitcoins are created. Bitcoin also creates a peer-to-peer network for exchange of the created coins. BitCoins provide measurable value, in that some retailers accept them in exchange for goods. In addition, some services allow for the exchange of BitCoins for government-backed currency, and is currently valued at over twenty U.S. dollars to one Bitcoin (BitCoinCharts, 2011). Even criminals have used these online anonymous coins for everything from purchasing drugs to fundraising for hackers (Chen, 2011; LulzSec, 2011). The monolithic currency approach of BitCoin as well as an overall limited distribution model has made many doubt the long-term viability of the solution.

2.6 Karma

As discussed in Chapter 1, Vivek Vishnumurthy proposes Karma (Vishnumurthy et al., 2003) as an economic framework for peer-to-peer networks. We review the specification of Karma further in Chapter 3. Here we review the various extensions to Karma.

Flavio Garcia built on Vishnumurthy's work in 2005 by specifying Off-line Karma (Garcia and Hoepman, 2005). The goal of Off-line Karma is to use electronic coins to conduct transactions without the need for contacting the bank set infrastructure. The off-line coins maintain a signed encrypted history of where the coin has been. At the time that the coin exceeds its expiration date, the bank set infrastructure is contacted to reconcile and mint a new e-coin. Fraud detection is enabled through blind signature technology that allows for the identity for someone who double spends the same coin. Off-line Karma greatly reduces the load and limits the complexity of peer-to-peer currency transactions. Off-line coins present a challenge in that fraud is only detected after it occurs. Significant fraudulent activity can occur before the automated detection is able to stop the impact.

In 2007 Sherman Chow built on the previous Karma work by defining Karma+ (Sherman, 2005). Chow's work addresses several weaknesses in the originally proposed Off-line Karma. Karma+ does not use coin expiration to trigger reconciliation. Rather the bank set infrastructure tracks coin ownership. After a transaction is complete, the new holder of the coin verifies its ownership of the coin using the bank set infrastructure. Karma+ utilizes the bank set concept but changes its function to track the ownership of coins rather than the bank balance of entities. Chow goes on to show the security of the bank set concept as implemented in his solution.

Clearly, there is a well-illustrated need for a secure online currency framework. To date there is no solution that is decentralized, reliable, applicable across all participant-based technologies, and secure. A combination of the reviewed technologies along with the introduction of new concepts enables a solution to meet the needs.

CHAPTER THREE

MultiKarma

We begin our solution by developing a public implementations of Karma. In describing our implementation, we explain the basic Karma protocol, identify numerous implementation oversights, and provide implemented solutions. Finally, we extend our solution to MultiKarma, which adds the ability to mint and trade multiple currencies over the decentralized banking infrastructure.

3.1 Original Implementation of Karma

Our contributions require an understanding of the original Karma proposal. We provide an overview of the basic structure and protocols to provide context for our improvements, additions and revisions. The theoretical nature of the proposal leaves significant work to achieve a functional system.

3.1.1 Banking Infrastructure

Karma uses a crowd-sourced banking infrastructure. Rather than having a central, trusted entity, Karma establishes trust by quorum. Bank set members are governed by a majority vote policy requiring `consensusresponses` > (`K`/2)+1 where `K` is the number of nodes in a node's bank set. The majority must also represent a consensus response and not just the most received response once the threshold of responses is met. For example, if during a transaction a node running on a Karma network with a `K`=3 receives only two responses from bank set members, these two responses meet the majority requirements. The node can consider the response valid only if the two received responses agree.

The value of `K` presents important trade offs. Due to the majority consensus policy, a larger `K` value makes the ability to poison responses or disable enough nodes to impact the network significantly more difficult. On the other hand, a larger `K` polynomially increases the number of messages sent on the overlay for transfer transactions. Query transactions only increase linearly in message count with regard to `K`. Chapter Four further tests and analyzes the cost/benefit proposition established by the value of `K`.

Every node in Karma must be able to identify the bank set for any given node. Nodes in Karma use bank set nodeIDs to enable communication to conduct a query or transfer transaction. This identification is accomplished through a one-way cryptographic hash known to all members of the network. The global nature of this function allows any node on the network to map the nodeIDs of bank set members. As a result, the true identity of a node (i.e., IP address) must be separate from the nodeID to protect against denial of service attacks and communication interception attempts. Secure overlay networks, described in Chapter 2, provide the abstraction necessary to separate a node's nodeID from its true identity.

The bank set solution provides two key attributes. First the bank set is completely decentralized, and no single entity can significantly influence the outcome of a transaction. Second, the solution allows stability through the natural churn that occurs in any peer-to-peer network. Karma replicates bank information to secondary nodes so that the data is maintained even if primary bank set nodes leave Karma.

3.1.2 Secure Routing

The robustness of routing in peer-to-peer networks depends on the cooperation of loosely-coupled untrusted nodes. As a result, peer-to-peer networks risk compromise if even a small portion of participants become malicious. Secure overlay networks address this risk by providing secure routing, which helps guarantee delivery of messages. Furthermore, a secure overlay must securely assign nodeIDs (to prevent selection of location in the overlay), securely maintain routing tables (to prevent route corruption) and securely forward all messages (to guarantee delivery of the message to the appropriate nodeID) (Castro et al., 2002). The Karma solution runs on a secure routing overlay network enabled by a DHT, such as Pastry. Utilization of the overlay network allows Karma to fully implement the abstraction of endpoints from the underlying transport network. Karma never utilizes IP addresses or endpoint identifiers other than the nodeID used by the routing overlay.

Overlay networks utilize routing algorithms that deliver messages to the node with a nodeID numerically closest to the destination nodeID. In the event the original recipient goes off line, this nearest-match behavior results in the next closest node receiving a message. Secure routing overlays further leverage closest-match routing to support a concept of *neighbor nodes*, which represent nodes logically closest to the nodeID of another node. The secure routing overlay provides a function call that returns a specified number of neighbor nodes. This ability allows for redundant information, such as account balance, to be stored at these neighbor nodes and enables the neighbors to take over communication in the event the originally responsible node goes off line. The proper leveraging of the overlay functions facilitates the implementation of a reliable and redundant banking solution.

3.2 Unaddressed Karma Implementation Issues

The original work on Karma focuses on two core functions: transferring currency and querying an account's balance. Significant changes to the original Karma proposal are necessary to arrive at our functional implementation. Vishnumurthy's work omits significant details required by a working implementation of Karma. Here we highlight the major changes needed to allow MultiKarma to function with the attributes and characteristics in the proposal.

The Karma proposal fails to provide an algorithm for mapping bank sets. Our solution utilizes the following algorithm for bank set mapping:

```
/**
  * Create a Vector listing the bank set nodeIDs for a given nodeID
  *
  * @param rice.p2p.commonapi.Id
  *             - the nodeID to map a bank set for
  * @return Vector<Id>
  *             - a Vector containing the Ids of the bank set
  */
  static public Vector<Id> generateBankset(rice.p2p.commonapi.Id nodeID) {
    Vector<Id> bankSet = new Vector<Id>();

    MessageDigest md = null;
    try {
      md = MessageDigest.getInstance("SHA");
    } catch (NoSuchAlgorithmException e) {

      throw new RuntimeException("No␣SHA␣support!", e);
    }

    md.update(nodeID.toByteArray());
    byte[] idDigest = md.digest();

    // Loop through Common.K to get the right number of nodeIDs for a bank set
    for (int i = 0; i < common.Common.K; i++) {
      bankSet.add(i, Id.build(idDigest));
      idDigest[]++;
      md.update(idDigest);
      idDigest = md.digest();
    }
    return bankSet;
  }
```

The function computes the `K` nodeIDs of a node's bank set by using *SHA1* hash of the given nodeID. The function then adds the resulting nodeID to an array of the bank set members and increments the nodeID value. The process continues with the function hashing the resulting value again to produce the nodeID of the next bank set member. This process continues for `Common.K` iterations to populate the bank set mapping.

While the original proposal requires the use of cryptographic signatures for balance verification, it fails to provide an implementation specification. MultiKarma uses the container `bankset.SignedBalance` class to store a cryptographically signed `bankset.Balance` object. The `bankset.Balance` contains the account balance along with a sequence number to help prevent replay attacks. Another attribute of the `bankset.SignedBalance` object is that it contains a signature verification function. This allows for simple signature verification by passing the public key stored in the bank set into the `bankset.SignedBalance.verifyBalance(Publickey)` function. Each bank set member stores the `bankset.SignedBalance` as the current balance in the account. This makes account fraud difficult as each time the balance value changes it must be accompanied by a new `bankset.SignedBalance` signed by the private key of the account holder.

3.2.1 Update Process

Our solution creates an update process that enables data replication and redundancy regardless of the number of currencies in an account. As original proposed, the update process transmits account changes, which are difficult to reconcile with multiple currencies. When a node joins the MultiKarma network, it must populate its local bank instance with the accounts for which it is responsible. Initialization of bank set account information begins by sending a `core.message.RequestAccountsMessage` to the neighboring nodes. The new node then assumes responsibility for accounts previously handled by one of its neighbors. Upon receiving the message, each neighbor compiles an `bankset.Account` listing that represents all the accounts in its bank repository. This method differs from original Karma, which only duplicates a few attributes (e.g., current balance) to newly joining nodes. The addition of multiple currencies causes the sending of a full `bankset.Account` to be more efficient and reliable for replication. Upon receiving the neighbor responses, the neighbor verifies the `bankset.Account` and adds it to the local bank repository.

A second function of the update process replicates account updates upon the completion of a currency transfer. When the transfer commits on a bank set member, the node creates an `core.message.UpdateMessage` containing the updated `bankset.Account`. This allows for full replication of the transaction history as well as the balances for each currency type. Next the node sends the `core.message.UpdateMessage` to `common.Common.U` of its neighbor nodes for account replication.

3.2.2 Bank Implementation

Several challenges need to be addressed through the bank set implementation. Karma ignores the implications of nearest-match functionality on bank set functionality. The nearest-match functionality in a secure routing overlay may result in a single node participating multiples times as a bank set member for the same node. Even more troublesome, nodes can participate in different roles within one transaction on the overlay network. Consider the case of nodeA transferring funds to NodeB where nodes A and B have bank set members with node IDs C and D, respectively. If the same node is responsible (due to the nearest-match routing) for nodeID C and D in the overlay, it would be participating as a member of both the sending and receiving bank set. This results in a need to maintain multiple bank states for a single node and demux calls to the local banking information.

We solve this issue by introducing an *idealID.* To help mitigate and simplify the policy when these situations arise, each node creates a separate `bankset.Bank` instance for each idealID received. Due to the bank set mapping algorithm leveraging

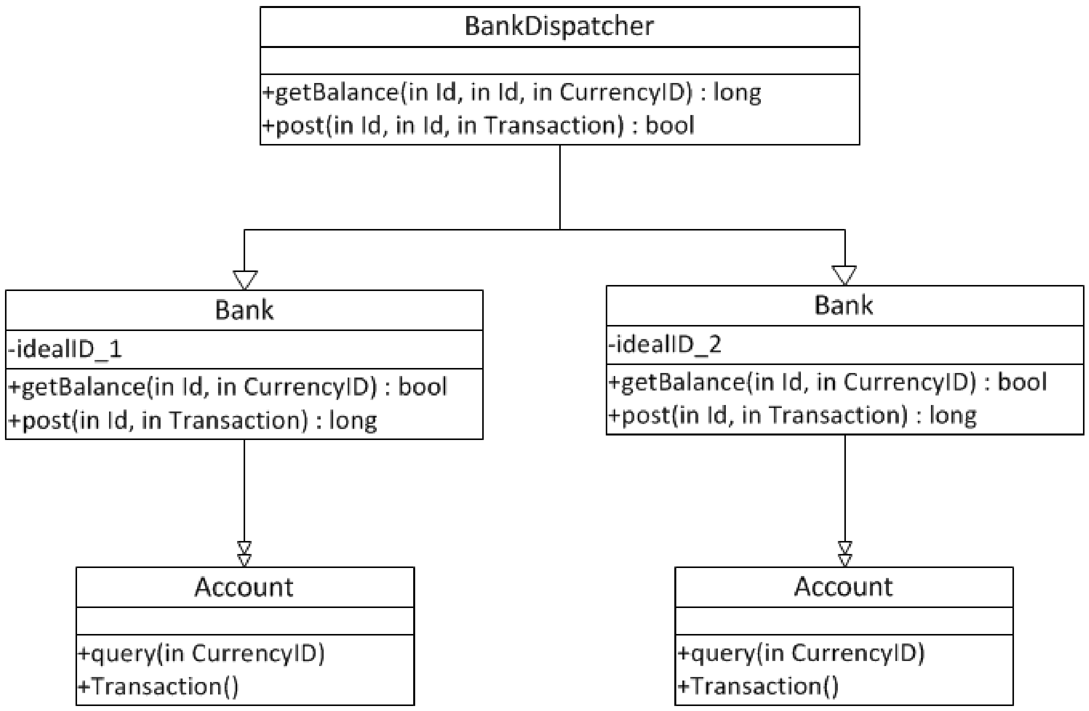
BankDispatcher
+getBalance(in Id, in Id, in CurrencyID) : long
+post(in Id, in Id, in Transaction) : bool
Bank
-idealID_1
+getBalance(in Id, in CurrencyID) : bool
+post(in Id, in Transaction) : long
Bank
-idealID_2
+getBalance(in Id, in CurrencyID) : bool
+post(in Id, in Transaction) : long
Account
+query(in CurrencyID)
+Transaction()
Account
+query(in CurrencyID)
+Transaction()

Figure 3.1: An instance diagram to illustrate the use of idealID to create a tiered banking infrastructure.

the full 140-bit nodeID space, messages can be sent to nodeIDs that are not active on the network. Because of nearest match behavior, the message receiver node ID seldom matches the message's destination node ID. The nodeID that is the intended destination, for the message, is identified as the idealID in MultiKarma. Each request to the local bank set member's bank is then handled by a `bankset.BankDispatcher` that maps idealIDs to `bankset.Bank` instances. This allows for a node to participate in multiple roles and be in different steps on the same transaction without bank balance or information conflict. Figure 3.5 illustrates the `bankset.BankDispatcher` and the use of idealIDs to create a tiering of banks.

MultiKarma requires replication of data to enable data resilience as nodes enter and leave the network. To enable efficient replication, a second change distinguishes between primary and secondary accounts within the `bankset.Bank`. Primary accounts are those for which the bank set member is currently responsible during a transaction.

If a node receives a `core.message.UpdateMessage` for a primary account, the disregard policy is enacted, and the message is not processed. Disregarding the updates, though not a perfect solution, prevents neighboring nodes who are members of the same bank set from causing primary to secondary churn within their `bankset.Bank`. Nodes only process query, transfer and new account transactions out of the set of primary accounts. If the `bank.Account` is located in a secondary set but needed to process a transaction, the account moves to the primary set.

Secondary accounts add an additional safeguard to account corruption by requiring that a node only accept updates for accounts located in secondary accounts. In the event a responsible neighbor node fails, the overlay routes messages to the next closest node. If a `bank.Account` is a secondary account when that message is received, it promotes to a primary account. `core.message.UpdateMessage` are the only type of messages that are posted to secondary accounts. This helps to prevent poisoning of a balance. Regardless of update type, the `core.message.UpdateMessage` contains a `bank.SignedBalance`, which contains cryptographic signature of the account owner. Having the signatures significantly increases the difficulty in creating fraudulent balances.

Rather than negative acknowledgments proposed in Karma, MultiKarma utilizes a timeout policy for transactions. The protocols within MultiKarma closely mirror implementations utilizing the *User Datagram Protocol(UDP)* for two-way, stateful communication. When a transaction starts, a timeout associates with the transaction. The timeout length varies depending on the type of transaction being conducted. For instance, a transfer transaction requires significantly more steps and messages to complete and, therefore, needs a different timeout value. If the transaction fails to complete before the timeout period, a notification removes the transaction from the list of active transactions within the node. This approach eliminates the need for acknowledgments and retransmissions; however unreliable networks may suffer difficulties in completing transactions. If a node receives a transaction and the timeout for that transaction has expired, the message is disregarded. As a result, negative acknowledgments are not required as the other processing nodes also timeout the transaction if they are unable to meet the majority consensus response policy in a timely manner. These policies, attributes and API objects create the framework for transactions to occur on MultiKarma.

3.2.3 Currency Valuation

As illustrated in the previous sections, the successful implementation of MultiKarma requires updating and revising multiple areas of the Karma specification. MultiKarma removes the currency valuation adjustment protocol which, as acknowledged in the original paper, uses an extremely network-intensive set of transactions to attempt to globally alter the value of the currency (Vishnumurthy et al., 2003). Due to the significant cost, the correction can only rarely be run, resulting in a flawed system for currency valuation management.

As describe in Chapter two, economists believe that free markets and currency competition naturally creates an economy that manages inflation and deflation. MultiKarma leverages its multiple currency framework to naturally control these factors. In addition, fine-grained control of currency valuation eliminates the need for entry computational hurdles. The wide adoption of user-generated multicurrencies reduces the need for a base currency. Users can transition value directly from one user currency to another without an escrow currency.

3.3 Enabling Multicurrency

Beyond creating a working implementation of Karma, one of the key goals for MultiKarma is to allow multiple currencies. To take the concept a step further, MultiKarma allows any user on the network to mint multiple currencies. We prefer to maintain the simplicity and elegance of the original Karma solution while still enabling these advanced features on the economic framework. This requires a number of core capabilities to enable the minting and trading of multiple currencies. Karma includes some of these basic requirements, such as transferring currency. Necessary functions like determining the amount of a given currency in circulation are not present in Karma. In addition, allowing any user to create a currency requires the ability to register the new currency into the economic framework. MultiKarma provides an elegant solution to this range of needs while not bloating the overall API or bank set requirements.

3.3.1 Function Changes and New Attributes

A user of MultiKarma must be able to identify the currency for each transaction; consequently MultiKarma introduces a currency identifier, `api.CurrencyID`, which includes two attributes: a nodeID (`rice.p2p.commonapi.Id`) and a currency sub ID(**short**). Inclusion of the currency sub ID allows for demuxing among a single user's multiple currencies, with each currency being identified through the unique nodeID and currency sub ID tuple.

```
/**
  * The constructor for creating a CurrencyID
  *
  * @param nodeID
  *            - the nodeID of the currency issuer
  * @param currency
  *            - the scalar identifier for this currency
  */
 public CurrencyID(Id nodeID, short currency) {
    this.nodeID = nodeID;
    this.currency = currency;
 }
```

All currencies on the MultiKarma network have an `api.CurrencyID` associated with them. A base currency exists on MultiKarma that no single user controls. An `api.CurrencyID` with a nodeID value of zero and currency value of zero represents the base currency. Using a base currency that is not user minted provides a more predictable valuation and limits fluctuation. A base currency may also reduce the complexity of currency exchange rates as only exchange rates between all user-minted currency to base currency is needed (as opposed to knowing all currencies pair exchange rates). While one could use government-backed currency for exchange, it was illustrated as inefficient earlier in Chapter 2.

Once currencies can be identified, the next step enables minting of currency. Rather than allowing users to mint numerous currencies that are never circulated, MultiKarma leverages the existing currency transfer process to mint a currency. As a result, a user who wishes to create a new currency must be willing to transfer some amount of that new currency to another user. When a transfer initiates with a new currency, the currency does not exist in the user's bank set unless the transaction successfully completes. The transfer function is thus altered to include the additional `api.CurrencyID` parameter.

The addition of multiple currencies introduces the need to know how much of a currency is in circulation. A simple change now allows a user to leverage the existing query function to determine the amount of a particular user's currency that is in circulation. A user may then assess valuation of a currency based on factors such as the amount of currency in circulation. MultiKarma alters the query transaction to support multiple currencies by also adding an additional parameter of a `api.CurrencyID`. For the transfer transaction, the minter of a currency increments its balance when sending currency to another user and decrements its balance when receiving a currency it minted. In effect, the minter's balance records the total amount of that particular currency in circulation.

As mentioned in Section 3.2.1, MultiKarma alters the update process to better accommodate multiple currencies. Rather than attempting to send deltas for `bank.Account` updates, MultiKarma sends the completely updated account object.

Due to multiple currencies, it is more reliable to send the full account object. As a result, `core.handler.UpdateHandler` need only handle one update process for both initial account replication as well as ongoing updates from completed transfers. Alterations to the core API function calls enable MultiKarma to support multiple currencies without forcing an entirely new set of functions.

3.4 Implementation Details

MultiKarma makes significant improvements to and defines previously unspecified policies for Karma to allow for a successful implementation of a distributed banking infrastructure. In addition, MultiKarma provides an implementation of the original Karma protocol. Here we present some details of implementation.

3.4.1 Common API

Secure routing overlay is a newer technology that continues to evolve. As a result, writing to the specific capabilities of any one overlay can leave the application orphaned as newer versions of the underlying network are released or abandoned. The Common API, discussed in Chapter 2, provides the flexibility MultiKarma needs to avoid this circumstance. MultiKarma uses only the function calls of the Common API. If Pastry (the secure overlay network that MultiKarma is currently utilizing) changes or the underlying implementation is superseded by a more secure overlay network, the Common API allows for a seamless transition to the newer technology.

Once the Pastry instance is established in the `Router` class, MultiKarma uses the objects and function calls of the `rice.p2p.commonapi` package. This package includes the `rice.p2p.commonapi.Endpoint(Endpoint)`, which is an abstraction of a secure overlay node object. The primary use for the `Endpoint` is to send messages on the overlay network. The `Router` class includes the required Pastry callback functions `deliver(rice.p2p.commonapi.Id, Message)`, `update(NodeHandle, boolean)` and `forward(RouteMessage)`, which are used to receive messages and network topology change information from the overlay. The unique characteristics of the secure overlay network, in conjunction with the base functions of sending and receiving messages from the overlay, enable creating a banking infrastructure.

3.4.2 Continuations

Generally, implementations handle network *I/O* in one of two ways: blocking or non-blocking. In blocking I/O, a node waits for a response before continuing with other tasks and execution of code. While this technique may be acceptable for simple programs, it does not map well to the complex implementation of MultiKarma. With non-blocking I/O, execution of other tasks continues, and the program is notified when I/O can proceed. This method of communication is significantly more appealing for MultiKarma.

In order to enable the MultiKarma API to provide non-blocking network communication, the solution leverages the continuation design pattern, *Callback*. When a function call requiring a later response is made to the MultiKarma API, the client includes a `Continuation` object that contains code to be executed at the time the network communication completes. `Continuation` may also include variables for response information that can be set to provide feedback on the transaction outcome. The package `api.continuation` included at the public API level of MultiKarma allows for the implementation of custom `Continuation` objects. MultiKarma includes `QueryContinuation` and `TransferContinuation` as two example implementations of `Continuation` objects.

3.4.3 Transactions in MultiKarma

MultiKarma enables the transferring of currency from one node to another as well as the querying of an account to determine its current balance. Prior to a node participating in these transactions, it needs to complete the fundamental step of creating an account with the MultiKarma system. First a node must establish an identity for use in all transactions. The identity consists of a 140-bit nodeID and a public/private cryptographic key pair. A new node (nodeA) must generate these two attributes such that they are linked as follows. NodeA generates and stores a public/private key pair. Next nodeA begins generating 140-bit nodeID values randomly until the public key and the nodeID value match in the lower `Common.N` digits. Similar to the `Common.K`, the value of `Common.N` is a globally defined constant linking NodeIDs with cryptographic keys that imposes a computational control on node creation. This is a feature that is implemented but not strongly relied on for security of the MultiKarma solution, as computational hurdles for entrance are generally quickly defeated.

After establishing its identity, nodeA must register the identity to MultiKarma. NodeA creates a `core.message.NewAccountMessage` and sends it out to the nodes that map as the bank set for nodeA. The message includes the generated nodeID and public key as well as a signed initial balance. The MultiKarma API enables this transaction with the function call:

```
/**
    * /** Constructor for MultiKarma instance
    * @param bindport
    *           - the local listening port for the MultiKarma instance
    * @param InetSocketAddress
    *           - the address of a bootstrap node already connected to the network
    * @param boolean
    *            - only true if this node will be providing bank set
        functionality only
    *           - primarily used for testing
    * @param String
    *           - the filename of an existing configuration file
    * @param String
    *            - the name of the node to appear in log files
    *          - helpful if multiple nodes are running in one JavaVM
    */
   public MultiKarma(int bindport, InetSocketAddress bootaddress,
         boolean bootstrap, String filename, String nodename)
```

Unless specified globally in the `Common.INITBALANCE`, the expected initial amount of currency is zero. Each member of the bank set processes the message and attempts to create an account for nodeA. If a bank set member is successful at creating the account, it replies to nodeA by updating the `NewAccountMessage` to reflect successful account creation. Once nodeA receives a majority consensus of affirmative messages from its bank set, it is considered registered to the network and able to conduct additional transactions. Once a bank set member adds a new account, it creates an `core.message.UpdateMessage` and sends it to its neighbors for replication and redundancy. The update process is covered later in this chapter.

3.4.4 Query

Conducting a query of an account is one of the two main functions MultiKarma provides. A query is a two-step transaction similar to account creation. Figure 3.2 gives a visualization of a query transaction. Consider a querying node (nodeA) and an account to be queried (nodeB). NodeA creates a `core.message.QueryMessage` with the nodeID of nodeB and `QueryMessage.type` field set to a value of `Common.QUERYREQUEST`. NodeA sends the message to the members of the bank set for nodeB. Each member of the bank set looks up the account balance for nodeB, inserts the value into the `QueryMessage`, changes the `QueryMessage.type` field to a value of `Common.QUERYRESPONSE`, and replies with its completed `QueryMessage` to nodeA. Any bank set members that do not have a balance for nodeB refrain from responding and query the overlay to create an account for nodeB in their local bank repository. The transaction completes upon the receipt of a majority consensus response at nodeA. NodeA then updates the status fields in the `Continuation` and calls the `Continuation.callback`() function to complete the transaction. In the event that majority consensus is not reached or a timeout occurs, the same `Continuation.callback`() is called, but the status fields are updated to reflect the failure.

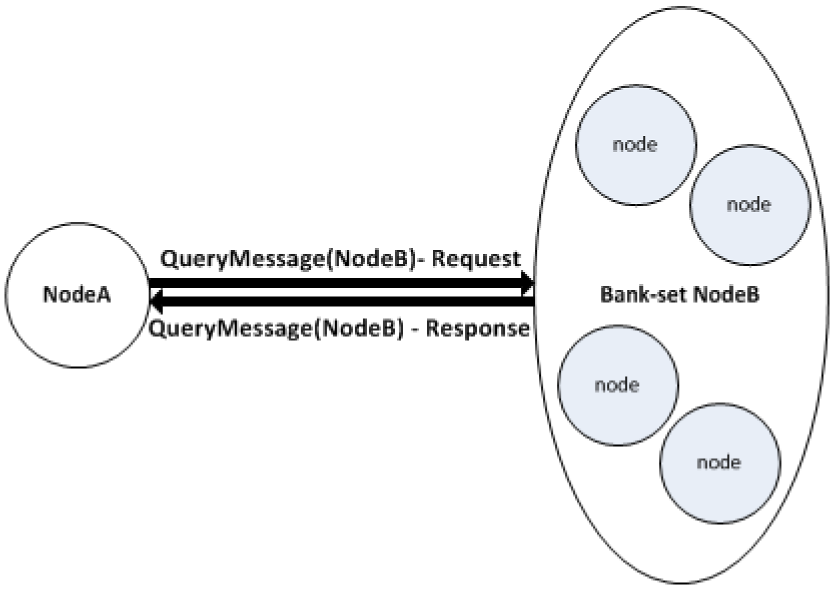

Figure 3.2: Query Transaction: NodeA queries for NodeB's account balance.

3.4.5 Transfer

Transferring of currency from one node to another constitutes the most complex transaction in Karma. A transfer transaction's message count complexity reduces to $O(K^2)$. This computational complexity of a transfer is the main reason there is a tradeoff consideration with the size of `Common.K`. Consider nodes A and B where Figure 3.3 shows the message exchange for the transfer transaction. This transaction mirrors the transfer protocol of the original Karma proposal.

NodeA begins a transfer transaction by creating a `core.message.TransferMessage`. The message includes the nodeID to send funds to as well as a `Common.TransID` (sequence number) and a `bankset.SignedBalance`, which is the balance following the completed transaction. In addition NodeA sets the `core.message.TransferMessage.type` field to a value of `Common.TRANSFER_ONE`. NodeA then sends the message to nodeB, and nodeB verifies the accuracy of the requested funds transfer. NodeB creates a second `core.message.TransferMessage` with the same fields as the original message but with values for nodeB and sets the `core.message.TransferMessage.type` field to `Common.TRANSFER_TWO`. The original `core.message.TransferMessage` is encapsulated in

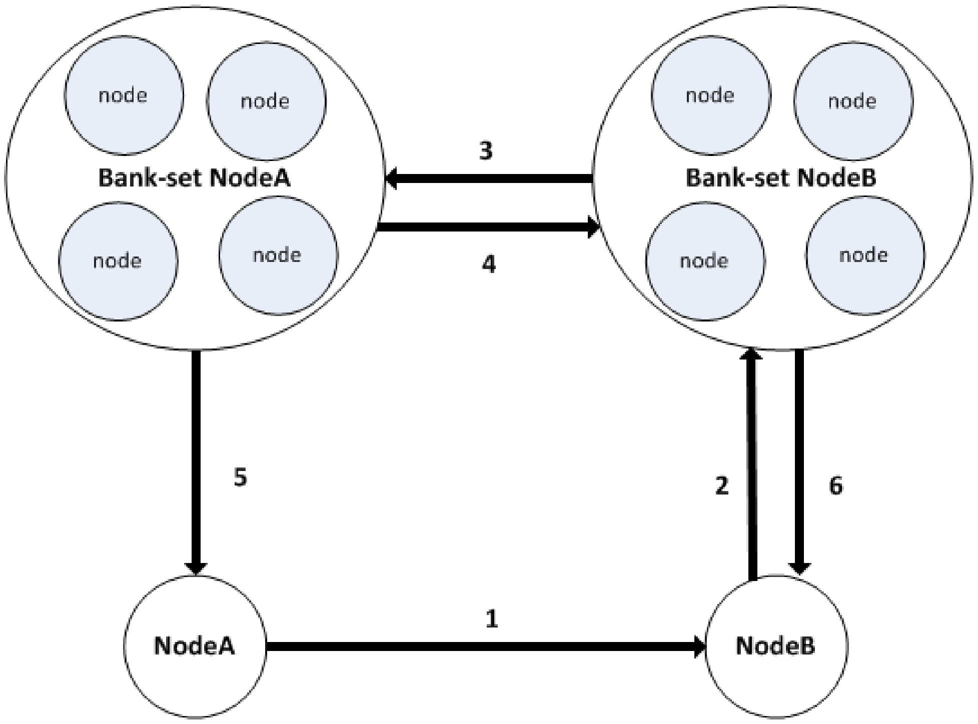

Figure 3.3: An example of the transfer protocol and associated messages.

the new message for reference throughout the transaction. NodeB then forwards the `core.message.TransferMessage` to its bank set (bank set nodeB). Each member of the bank set validates the cryptographic signature on the `bank.SignedBalance` provided by nodeB and validates the accuracy of the balance value following completion of the transaction and the accuracy of the `common.TransID`. If all the fields are accurate, the bank set member updates the status fields in the `core.message.TransferMessage` to `Common.TRANSFER_THREE` and forwards the message to each member of the bank set for nodeA. Each bank set member for nodeA then validates the same fields at the bank set of nodeB. If the fields are valid and a majority consensus has been received, the bank set for nodeA commits the transaction to the account of nodeA. Each bank set member then updates the `core.message.TransferMessage` with a type of `Common.TRANSFER_FOUR` and forwards to each of the bank set members for nodeB. Each bank set member for nodeA then completes the step of notifying nodeA that the transaction is complete. This is done by forwarding the `core.message.TransferMessage` to nodeA with the status field set to `Common.TRANSFER_FIVE`. Once nodeA receives a majority consensus, it commits the transaction to its local copy of its account. The receiving bank set (nodeB) waits for a majority consensus of responses from the bank set of nodeA and then each member commits the transaction to nodeB's account. Upon completion each nodeB bank set member forwards the `core.message.TransferMessage` to nodeB with a status field of `Common.TRANSFER_SIX`. Once nodeB receives a majority consensus from its bank set, it commits the transaction and completes the transfer transaction. When the receiving and sending nodes receive majority consensus, they execute the `api.Continuation` associated with the transaction. If a node is unable to verify any of the fields during a step in the transaction, it institutes a disregard policy and does not send on the `core.message.TransferMessage`. It is not possible for transactions to be processed out of order due to the cryptographic signatures on the account balance. If a bank set member believes it is out of sync, it may query the other members of the bank set to attain a current balance using the majority consensus.

3.4.6 Define the MultiKarma API

MultiKarma adds the functionality of multiple currencies without increasing the API functions. First MultiKarma extends query function call to support multiple currencies by adding the `api.CurrencyID` as follows:

```
/**
    * /** Query for MultiKarma
    *
    * @param rice.p2p.commonapi.Id
    *           - the Id of the account for query a balance for
    * @param CurrencyID
    *           - the currency to query for
    * @param Continuation
    *            - code to be executed once the transaction is completed
    *           - this code will be called even if the transaction fails
    */
```

```
public void queryAccount(rice.p2p.commonapi.Id id, CurrencyID currency,
        Continuation cont)
```

Second MultiKarma updates the transfer function to support multiple currencies. Like query, transfer now includes an additional parameter, `api.CurrencyID`, as follows:

```
/**
    * /** Transfer for MultiKarma
    *
    * @param long
    *            - the amount of currency to transfer
    * @param CurrencyID
    *            - the currency type to transfer
    * @param Continuation
    *             - code to be executed once the transaction is completed
    *            - this code will be called even if the transaction fails
    * @param rice.p2p.commonapi.Id
    *            - the Id that the funds will be transferred to
    */
    public void send(long amount, CurrencyID currency, Continuation cont,
         rice.p2p.commonapi.Id dest)
```

3.4.7 Dispatchers, Handlers and Messages

Message objects are the fundamental way communication occurs with MultiKarma. All messages in MultiKarma derive from `core.message.TaggedMessage`. Figure 3.4 provides a visualization of the tiered approach implmented in MultiKarma. This object only contains a single field, `core.message.TaggedMessage.tag`, which is used to demultiplex to the variety of sub-typed messages. `core.message.TaggedMessage` also extends `PastryMessage`, which allows for all MultiKarma messages to traverse the Pastry overlay. In the event that the underlying overylay network needs to change, `core.message.TaggedMessage` inheritance can be altered to the appropriate new overlay message class. Layering the messages allows for them to be treated as individual envelopes. Each step of the demultiplexing process only needs to check one attribute to enable a dispatch decision.

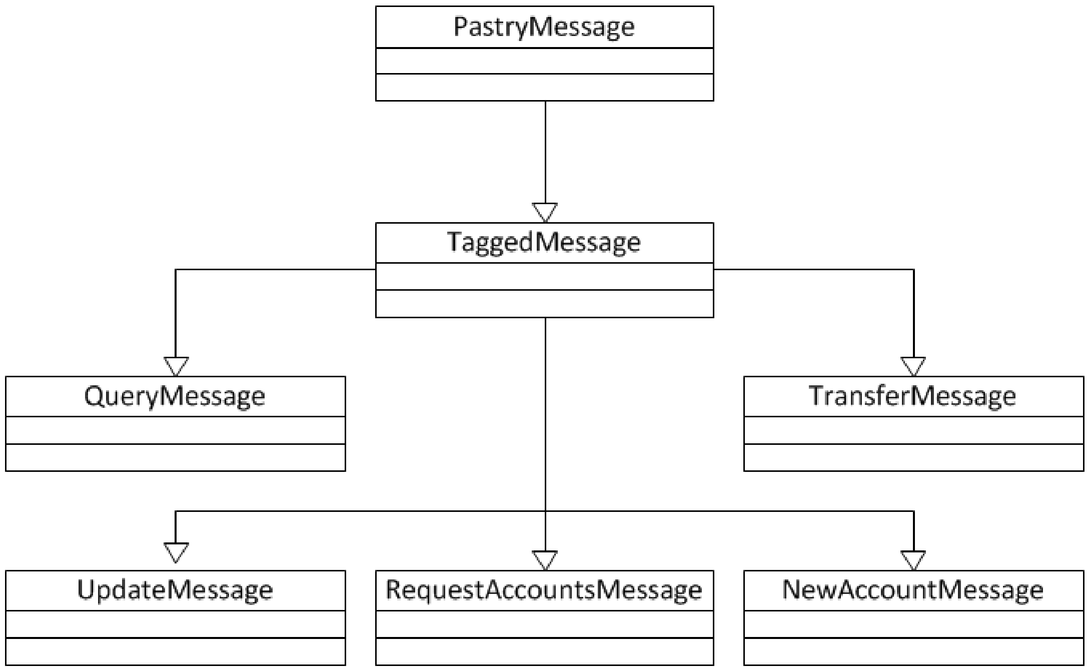

Figure 3.4: A class diagram for the layered message approach in MultiKarma.

The MultiKarma node processes messages in a layered approach. `core.Router` hands all messages to `core.dispatcher.TypeDispatcher`, which evaluates the type of message received and passes the message to the appropriate type specific dispatcher. Each of the type-specific dispatchers (`core.dispatcher.QueryDispatcher`, `core.dispatcher.TransferDispatcher` and `core.dispatcher.NewAccountDispatcher`) either processes the message if the transaction is simple or passes the message along to a handler. If the message is the start of a new transaction, a new type specific `core.handler.HandlerTemplate` instantiates at hand-off. Figure 3.5 illustrates this layering approach used for this message processing.

The dispatcher hands messages that represent continuations of in progress transactions via demux to the existing type specific handler. Since update transactions do not require state maintenance, `core.dispatcher.TypeDispatcher` hands `core.messages.UpdateMessage` straight to a `core.handler.UpdateHandler`. For all other transactions, handlers maintain state information. Due to the majority consensus requirement, each transaction must track the number of responses that are in agree-

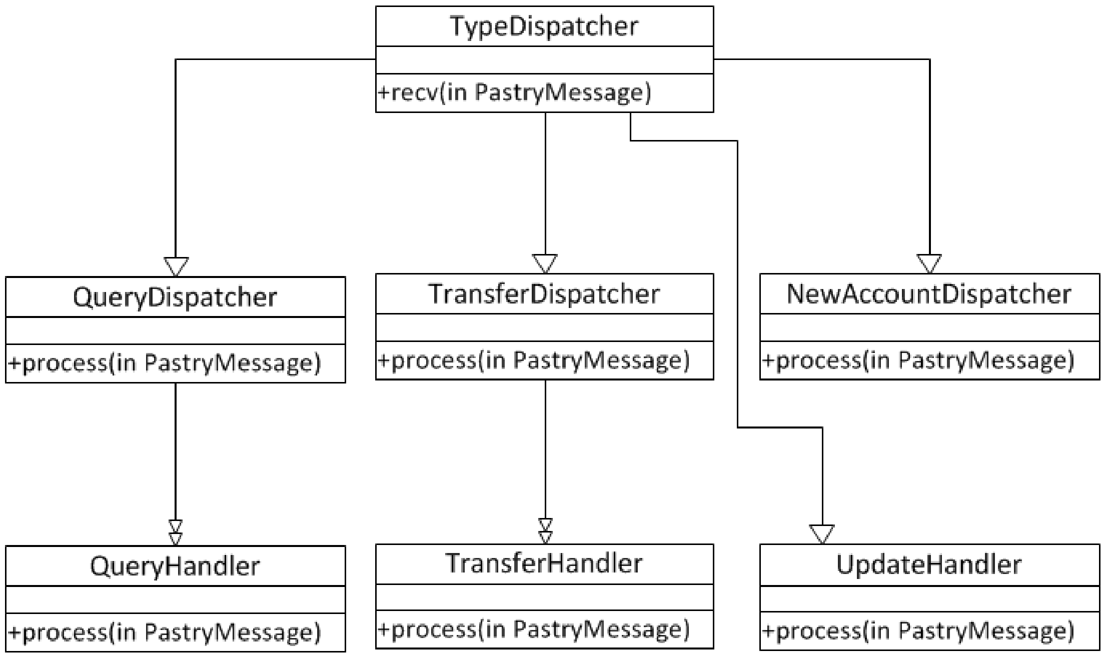
TypeDispatcher
+recv(in PastryMessage)
QueryDispatcher
+process(in PastryMessage)
TransferDispatcher
+process(in PastryMessage)
NewAccountDispatcher
+process(in PastryMessage)
QueryHandler
+process(in PastryMessage)
TransferHandler
+process(in PastryMessage)
UpdateHandler
+process(in PastryMessage)

Figure 3.5: A class diagram to illustrate the relationships and information flow between dispatchers and handlers.

ment. When a handler processes a message that meets the majority consensus, the transaction completes, and the handler removes itself from the list of handlers in its dispatcher. Handlers also maintain timers for transaction timeouts. Upon instantiation handlers set a timer based on the timeouts available in the `common.Common` class. If the timeout is exceeded, the handler wakes up and removes itself from the list of available handlers, effectively ending the transaction. It also executes any `api.continuation.Continuation` code with the appropriate status settings.

Building on the work of Vishnumurthy, MultiKarma creates a fully decentralized multicurrency framework that allows any user to create a currency. Our solution provides solutions to the unanswered questions of the original proposal. Chapter 4 evaluates MultiKarma's ability to scale and message complexity.

CHAPTER FOUR

MultiKarma Evaluation

Due to the lack of a public implementation, a Karma implementation has not been evaluated on critical attributes like scaling. MultiKarma provides the first opportunity to evaluate the ability of a consensus-based banking infrastructure to provide scalable and reliable service. We establish a simulation-based evaluation, determine metrics, and analyze the results to determine the scalability of MultiKarma with respect to our criteria.

4.1 Simulations

We begin our simulation by constructing an overlay network of Pastry nodes. A special "bootstrap" node is specified to each newly joining node to determine where to connect in the overlay. After initial connection, the overlay takes responsibility for improving the physical performance of overlay routing by evolving the mesh such that proximity in the overly more closely reflects network topology. All mesh nodes are capable of routing messages to other nodes. All nodes in the mesh are capable of participating as a bank set member.

To simulate transactions, some subset of the mesh nodes execute simulated account holders called actors. Actors constitute twenty percent of the total network size during a simulation. Actor actions and times are randomly selected from a uniform distribution within a specific range. The script is as follows:

```
/** 0 = Query, 1 = Transfer */
    while( continue ){
        //sleep for a random period of time less than the maximum sleep time
        //for our texting max sleep is 30000ms
        Thread.sleep(random%maxsleep);
        //select a transaction type
        transactiontype = random%2;
```

```
//select a traget to conduct the transaction with
//same node allowed for query and transfer (no impact on transfer)
targetactor = targetNodeIDs.get(random%totalactors);
/**  currencytype
      All users start with 100 base currency
      for transfer is base or sender currency
      for query is base or target node currency
*/
currencytype = random%2;
//transfers need an amount of currency to transfer
if( transactiontype == 1){
   amount = random%10;
}
conductionTransaction();
```

An actor sleeps and then executes a transaction. The actor then goes back to sleep for a random period of time between 0 and max sleep time. A simulation ends when the max execution time is reached. At that time, a flag is set on each actor to notify them to end the simulation and terminate the MultiKarma instance.

4.2 Evaluation Metrics

We measure several metrics to understand the performance and scalability of MultiKarma. Users are impatient, and MultiKarma needs to complete transactions with minimal delay. For instance a transaction time of more than a few seconds results in a user waiting and possibly not using MultiKarma. In addition, MultiKarma has broad application and needs to be able to scale with significant usage.

First we evaluate the total time for a transaction to complete to understand the delay a user may experience. The transaction time is measured from the point the node instantiates the transaction callback until the point the transaction callback executes. Second, we evaluate message count for transfer, query and update. Message count impacts the scalability of MultiKarma through bandwidth utilization and message processing delay. We evaluate message count in a close form solution since it does not vary between occurrences. Finally, we measure the size of the messages traversing MultiKarma for queries, transfers and updates. During each step of the transaction, nodes record the size of a message to a log prior to sending it over the network. Evaluating these metrics allows for an evaluation on the scalability of MultiKarma.

4.3 Experiments

Our experiments use actors to capture measurements for the metrics we wish to evaluate. Each simulation has a set `K` and network size. A simulation executes for a total of 5 minutes. The simulations run through five iterations. To eliminate network propagation delay and evaluate only MultiKarma's transactional complexity, all nodes for the simulations run on a single computer (may introduce CPU wait). Each actor also records the total message size in a log before sending the message onto the network.

As described earlier, the global value `K` determines the bank set size. The value of `K` is a global value that is difficult to change once the MultiKarma instances are deployed. We vary the size of `K` and network size as these are the most likely to impact our metrics. `K` ranges from five to nine. While this presents a small range of values, it provides a significant range in transactional complexity. We range the network size from 20 to 60 nodes. This size range allows for analysis of trends that may occur as the MultiKarma network grows. The `MultiKarmaSimulation` program runs the testing simulations and accepts three variables: network size, max execution time and max sleep time. The size of `K` is specified in `Common.K` and changes for each testing iteration.

4.4 Results

Our results show the impact of varying `K` and network size on our metrics transaction time, message count and message size. Each `K` and network size combination has a minimum of 50 samples for each transaction type (e.g., query and transfer).

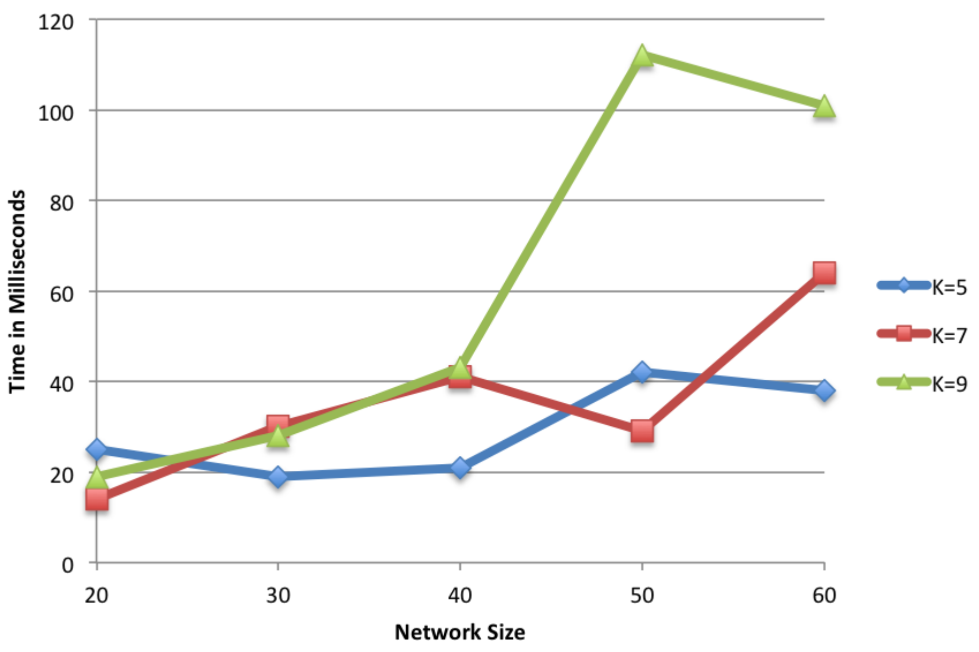

Figure 4.1: Query average execution time with varying K and network size.

Since the number of actors (e.g., actors constitute twenty percent of network size) varies with network size, samples per transaction type increase to over 200 for each K at a network size of 60.

4.4.1 Transaction Time

We begin our analysis by evaluating the impact of K and network size on query time. As a baseline, a query on a network with twenty nodes, four of which are actors, and a K size of five experiences a mean execution time of 25 milliseconds. All time measurements are actual elapsed time, not computation or network time.. Referring to the results in Figure 4.1, one can see that query times for small networks stay relatively constant, only varying by a few milliseconds and hovering between 20-40 milliseconds. Larger networks with larger K values only experience an increase to around 60-100 milliseconds for mean execution time. Queries occur in parallel but messages leaving MultiKarma process serially. As a result a larger K requires more message transmissions causing sending delay. In turn, message count complexity can

Table 4.1: Query min and max execution time (in milliseconds) for varying `K` and network size.

	Network Size				
K Size (Min, Max)	20	30	40	50	60
5	(6, 633)	(7, 338)	(9, 44)	(11, 662)	(11, 2278)
7	(7, 36)	(8, 474)	(10, 757)	(12,78)	(14, 1187)
9	(9, 245)	(11,377)	(14,808)	(15,1824)	(19, 1956)

impact transaction time. Network size can also have an impact, due to it increasing the number of routing hops to reach a destination node. This can be compounded by Pastry only minimally interconnecting nodes based on geography. As a result as `K` increases in size, the probability of encountering a node on the pathological path increases. Since all tests were conducted on a single four core CPU, it is possible that larger network sizes require more CPU wait time. Query represents a fast transaction, and a small amount of CPU wait or delay can skew the graph as represented with a `K`=9 and network size of 50.

Table 4.1 displays the minimum and maximum execution times for each of the test sets. The minimum values illustrate that when an ideal transaction takes place that results in a minimum execution duration, we see a general upward trend in time as `K` and network size increase. The table also illustrates that situations arise that can greatly delay the completion of a transaction. This could be due to CPU wait time on the single system being used for testing and appears to generally increase as the network size increases. In summary, query appears to be a simple enough transaction that varying network or `K` sizes has limited impact on transaction performance.

Next we evaluate execution time for transfers. As discussed earlier with query, message count complexity can affect transfer times. Transfer is a transaction with a message count complexity of $2K^2 + 3K + 1$ or $O(K^2)$. Figure 4.2 illustrates that

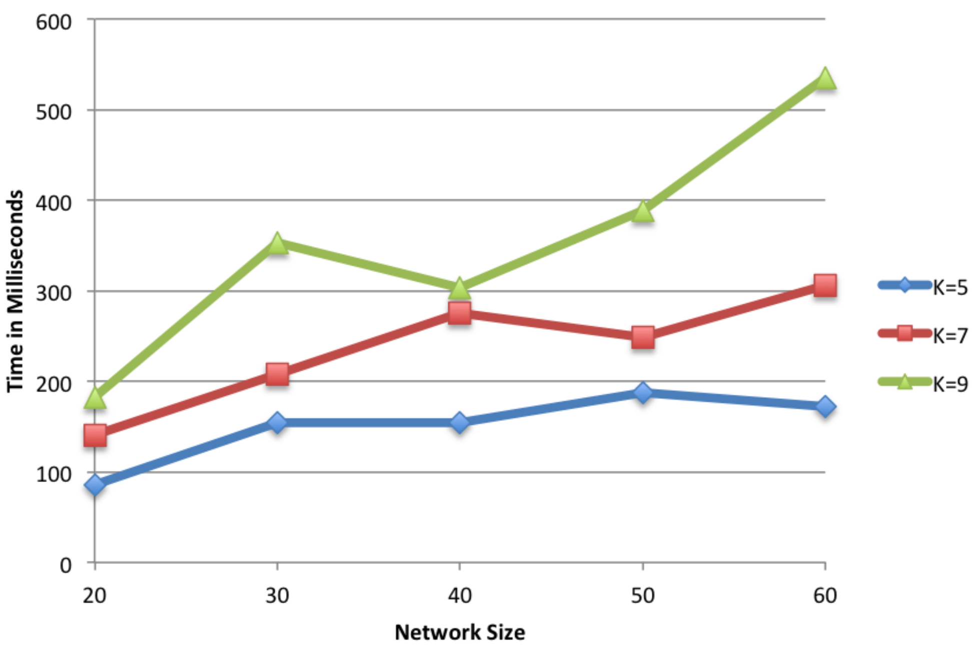

Figure 4.2: Transfer average execution duration with varying `K` and network size values.

both network size and the size of `K` impact the duration of transactions. Conducting transfers on a relatively small network of twenty nodes with a `K` of 5 experiences a mean execution duration of 86ms. Evaluating the same size network but increasing to `K` of 9 increases the mean execution duration to 183ms. While over a two-fold increase, this time is well within the range of acceptable for a complex transfer. The increase does highlight the importance of selecting a reasonable `K` value. As network size increases, execution time for transfer increases. MultiKarma's utilization of a mesh routing network results in additional routing hops as the network size increases. Overlay routing tries to minimize this delay, but we clearly see some delay increase at the same `K` value. For instance, a `K` value of seven increases the total transactions time from 141ms up to 535ms, for network sizes of twenty and sixty, respectively.

Table 4.2 presents the minimum and maximum transfer execution times for `K` and network size. Similar to query, transfer minimums, representing the ideal duration time, have an upward trend as either `K` or network size increase. While the

Table 4.2: Transfer min and max execution time (in milliseconds) for varying `K` and network size..

	Network Size				
K Size (Min, Max)	20	30	40	50	60
5	(33, 325)	(43, 572)	(56, 658)	(55, 1518)	(59, 802)
7	(38, 540)	(55, 1032)	(65, 915)	(70,1073)	(84, 2140)
9	(58, 668)	(73,1802)	(76,1494)	(95,1921)	(101, 3273)

upward trend for transfer mean and minimum is not significant, we see significant outliers with high maximum execution times. At the worst, we see a transfer with a K=9 and network size of 60 taking over 3 seconds to complete. As described earlier, a network size of 60 with 12 actors likely plays a role in this increased delay and burst in maximum transfer times due to CPU wait and routing hops. Regardless of network size, `K` appears to have an impact on the completion time of both queries and transfers.

Given this evaluation, it seems reasonable to select a modest `K` to increase performance and allow for the update process to provide replication for redundancy. The goal of maintaining a total transaction time of less than one second for the most complex transaction of transfer seems reasonable. Next we evaluate message size and count to understand additional metrics that may affect performance.

4.4.2 Message Analysis

Our evaluation of messages on MultiKarma concentrates on two metrics: message count and message size complexity. We review these metrics in isolation and then in combination to evaluate their total impact on the network. The evaluation of these metrics occurs for each transaction type: query, transfer and update.

Queries provide a simple message count complexity of $2K$. The querier must send a message to each bank set member and receive a corresponding response.

Consider a nodeA who queries the account of nodeB. NodeA sends a request to each of the `K` bank set members for nodeB. Each bank set member responds to the request with a query response sent to nodeA. So for any query, the total number of messages exchanged is linear with the size of $O(K)$. Messages for a query constitute a set number of fields that vary little in size. In reviewing query message size from the data captured during simulation, the mean message size for a query is 720 bytes. The variation in message size regardless of network size or `K` is less than 100 bytes. This stability of size results in query messages having a predictable impact on the network. Queries and their associated messages do not present a decision metric for influencing the size of `K`. However, queries do illustrate a highly efficient algorithm that minimizes network utilization and transaction time to achieve balance retrieval.

Next we review the message count for transfers. Transfer represents the highest message count complexity transaction on MultiKarma, $O(K^2)$; therefore, transfers present a significant potential for impact to the network. This count can quickly result in over seven hundred total messages sent for a reasonable `K` value of 19. If the secure overlay network MultiKarma utilizes meets the securing routing specification, it becomes non-trivial to compromise a majority of the bank set members. Our assumption of a "secure" overlay network must hold true for MultiKarma to maintain accurate banking functionality.

Our focus now shifts to the size of transfer messages. Similar to query messages, transfer messages have a fixed number of fields that vary little in size. In reviewing the captured simulation data, transfer messages ranged from approximately 1800 to 2400 bytes, with messages typically close in size to the ends of the range. Recall from the transfer protocol that the second step involves the receiving node encapsulating the original transfer message into a new transfer message. This account for the size variation and the two distinct sizes we captured during simulation.

As we combine the two analyses for transfer messages, we can see a greater impact to the network. Consider a network with a `K` value of 9. The total transaction requires 418 KB of data to traverse the network. If we increase the value of `K` to 19, the total bytes of data jumps to 1716 KB. While these numbers are minuscule with today's broadband speeds, we may see the day when a bank set size of 50 is realistic for financial transactions. In this case, MultiKarma would require 5151 messages with a size totaling 11332KB.

To conclude our message metric analysis, we review update messages. Unlike the minimal implementation changes to `NewAccountMessage`, `QueryMessage`, and `TransferMessage`, `UpdateMessage` alters due to multiple currencies. The original Karma specification only sent account deltas for each update message when a transaction posted. This approach works for only a single currency. MultiKarma, on the other hand, stores all of a user's currencies in one account on each bank set member. While a new node joining the overlay receives account replication from its neighbors, scenarios exist where the delta solution results in updates being received for a node and the bank set member not having account information to which to apply the delta. Consider the case where NodeB is currently a bank set member for nodeA and has replicated its account information to nodeC who can take over at nodeB's failure. Now consider nodeD, a neighbor to nodeC. If nodeC and then nodeB were to fail, nodeD would be left as the bank set member for nodeA without proper account information. To help resolve this issue, each time an `UpdateMessage` occurs, the message contains the entire contents of the account object. In turn, the situation of receiving a delta without appropriate account information no longer occurs.

We now review the message count complexity and size for update messages. Each time a node sends out an update message, it is destined for its `Common.U` neighbors. This protocol results in a linear message complexity and is only a one way communication as the sending node does not get a response for sending the update. As a result `UpdateMessages` message count complexity presents little value as a decision metric for MultiKarma. `UpdateMessage` does vary in size based on the number of currencies contained within an account. As described earlier, MultiKarma sends full account information with every update. In reviewing our simulation logs, update messages start at a modest size of 1900 bytes. Towards the end of the simulation as each node maintains multiple, user-minted currencies, we see the update messages grow to an approximate size of 10,000 bytes. Fortunately, the update protocol only sends messages to the `Common.U` neighbor nodes and not the nodes bank set limiting the overall impact.

From our analysis, transfers constitute the greatest impact to scalability of MultiKarma. When the MultiKarma network is established, one must select the global `K` value. This decision must be made considering the total execution duration of a transfer. Message size is not a critical criteria for performance of the MultiKarma solution. While it is unclear how many currencies the average user will maintain, the number is likely to be far less than would cause any impact to the network. If users are regularly maintaining thousands of currencies, the sending of full account objects needs to be reevaluated to determine if the benefit of guaranteed replications is still valuable.

CHAPTER FIVE

Conclusion

Peer-to-peer and other participant-based technologies depend on the quality contribution of their users to produce appealing resources; consequently such system must incentivize participation to maintain acceptable levels of quality resources for survival. Incentive schemes for users to contribute are lacking or problematic in many systems, resulting in a need to account for consumption and contribution. Solutions for tracking contribution and consumption vary. They include reputation systems, which provide a metric of trustworthiness but fall short as they can be exploited for short-term gain. Synchronous tracking solutions, such Gnutella's tit-for-tat, which only allows consumption with simultaneous contribution and lacks quality control. Asynchronous tracking systems, such as SWIFT, enable contribution credit but are designed to service one group of participant based technologies. Clearly the ideal solution is a digital currency system that allows asynchronous contribution/consumption and portability.

There have been many attempts to create a universal digital currency system. Generally the most widely known solution is the e-wallet solution from PayPal. We have also seen niche currency solutions, such as Facebook credits or Warcraft gold. Sadly these solutions are centralized, lack portability and granularity and are laden with transaction costs (i.e. primarily due to being tied to government currency). Current currency solutions only containing one currency is another limitation. Friedrich Hayek illustrates that economies with multiple currencies allow for free markets and economic competition. Our solution solves these limitations.

5.1 MultiKarma

We have created a working MultiKarma implementation that extends the original Karma proposal by providing 1) solutions to unanswered implementation specifications and 2) extensions to support multiple user-minted currencies. Our solution publishes an API that provides the flexibility to leverage the solution on any participant-based technology, resulting in value portability. The addition of multiple currencies allows free markets to flourish with currency competition. Finally MultiKarma frees users to conduct transactions and exchange currencies without transaction fees.

Our implementation extends the banking infrastructure to account for the nearest-match routing behavior of a routing overlay. We introduce the concept of an idealID for bank instance demuxing. MultiKarma provides a function for mapping bank sets. We redesign the update process to allow for multiple currencies enabled through the introduction of a currencyID. MultiKarma utilizes Java to provide platform independence. The solution further enables its resilience by utilizing the Common API to enable transitioning of the underlying secure overlay routing solution. MultiKarma's design allows for it to endure changing technologies and continue providing a distributed banking infrastructure.

We evaluate MultiKarma's ability to scale. Our experiments measured the transaction duration, message count and message size. We vary the size of the network as well as the size of `K` to determine their affect on the metrics. While there is an increase in transaction time as both variables scale up, their impact can be mitigated. `K` selection needs to balance transaction time and bank set security. Network size increase affecting transaction duration is likely a function of the single system testing environment and not of a larger scaling issue. Message count is directly functional to the size of `K` for query and transfer and `U` for updates. Both query and transfer messages remain constant in size irrespective of K and network size. However update messages linearly increase in size with the number of currencies stored in the account. We believe that selection of a reasonable size `K` allows for MultiKarma to scale to large sizes without increasing the metrics we evaluated to user unacceptable levels.

5.2 Future Work

While a solid solution for currency exchange and minting, MultiKarma would benefit from additional features. The original Karma proposal covers the importance of transaction negotiation but leaves that functionality as well as reputation metrics to external systems (Vishnumurthy et al., 2003). Like the original Karma proposal, MultiKarma does not address reputation and transaction negotiation services. Reputation services provide trust metrics to judge the reliability and quality of an entity. Transaction negotiation services allow for the location and pricing of resources. While we consider these orthogonal issues, they are nonetheless necessary in a fully functional market system.

Online systems involving value exchange benefit from reputation metrics. Two areas of MultiKarma provide opportunities for reputation metrics. First, buyers and sellers in MultiKarma could benefit from a reputation measure. Users of MultiKarma would utilize reputation to judge the trustworthiness and reliability of another participant. Second, traders of currency may also benefits from reputation knowledge of a currency owner. This helps a currency user to evaluate the stability and reliability of a currency on MultiKarma.

Another improvement to MultiKarma is the addition of a framework for transaction negotiation. MultiKarma's design allows adoption across a wide array of participant-based technologies. Such an extension to MultiKarma could provide services such as resource location, resource cost, and currency exchange. One challenge to development of this service is providing a framework generic enough to address all negotiation needs.

Another extension to MultiKarma is the ability to withdraw currency from an account and store the value as a digital coin to enable exchange free of bank set interaction (so called offline currency). As discussed in Chapter 2, Flavio Garcia proposed converting Karma to a solution for reconciling offline coins. We prefer to see a hybrid solution where currency can be stored in banks and then withdrawn for offline use. In addition to providing bank set free transactions, offline coins allows for some level of anonymity, by utilizing blind signatures, similar to traditional currency.

MultiKarma introduces the unique concept of allowing any user to mint currencies. We would like to see this economic concept adapted and explored in existing digital currency solutions, such as BitCoin. Previously, we highlighted the advantages of multiple currencies in economic theory and practice. It is only practical that digital currency systems would experience the same benefits.

BIBLIOGRAPHY

Adar, E. and B. A. Huberman (2000). Free riding on gnutella.

Beenz (1999). www.beenz.com. http://web.archive.org/web/19990508193610/http://www.beenz.com/.

BitCoinCharts (2011). Bitcoin charts / markets. http://bitcoincharts.com/markets/.

Castro, M., P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach (2002). Secure routing for structured peer-to-peer overlay networks. In *ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pp. 299–314.

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

Chen, A. (2011). Underground website lets you buy any drug imaginable. http://www.wired.com/threatlevel/2011/06/silkroad//.

Christin, N., A. S. Weigend, and J. Chuang (2005). Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, pp. 68–77.

Clarke, I., O. Sandberg, B. Wiley, and T. W. Hong (2001). Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop On Designing Privacy Enhancing Technologies: Design Issues In Anonymity And Unobservability*, pp. 46–66. Springer-Verlag New York, Inc.

Dabek, F., B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica (2003). Towards a common api for structured peer-to-peer overlays. In *International Workshop on Peer-To-Peer Systems*.

Damiani, E., S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante (2002). A reputation based approach for choosing reliable resources in peer-to-peer networks. In *In 9th ACM Conf. on Computer and Communications Security*.

Druschel, P. and A. Rowstron (2001). Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, pp. 75–80.

Entertainment, S. O. (2011). Station exchange: The official secure marketplace for everquest ii players. http://stationexchange.station.sony.com/livegamer.vm.

Flooz (1999). www.flooz.com. http://web.archive.org/web/19991013032840/http://flooz.com/.

Garcia, F. D. and J.-H. Hoepman (2005). Off-line karma: A decentralized currency for peer-to-peer and grid applications. In *Proceedings of the 3rd Applied Cryptography and Network Security*.

Giesler, M., M. Pohlmann, M. Giesler, and M. Pohlmann (2003). The anthropology of file sharing: Consuming napster and as a gift.

Gupta, M., P. Judge, and M. H. Ammar (2003). A reputation system for peer-to-peer networks. In *Proceedings of NOSSDAV*.

Hayek, F. A. (1978). Denationalisation of money - the argument refined: Second (extended) edition.

Heeks, R. (2008). Current analysis and future research agenda on "gold farming":real-world production in developing countries for the virtual economies of online games.

Hoffman, K., D. Zage, and C. Nita-rotaru (2009). A survey of attack and defense techniques for reputation systems.

Horne, B., B. Pinkas, and T. Sander (2001). Escrow services and incentives in peer-to-peer networks. In *In EC*, pp. 85–94.

Jambool (2010). Social gold. http://web.archive.org/web/20101203194330/http://www.jambool.com/.

Jevons, W. S. (1876). *Money and the Mechanism of Exchange*. Library of Economics and Liberty.

Kamvar, S. D., M. T. Schlosser, and H. Garcia-Molina (2003). The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*.

Keromytis, A. D., V. Misra, and D. Rubenstein (2002). Sos: Secure overlay services. In *Proceedings of ACM SIGCOMM*, pp. 61–72.

Liang, J. (2004). Understanding kazaa.

LulzSec (2011). Lulzsec versus fbi affiliates + whitehats - pastebin.com. http://pastebin.com/MQG0a130.

Moncash (2011). http://www.moncash.com/.

Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/.

Peng, D., W. Liu, C. Lin, Z. Chen, and X. Peng (2008). Enhancing tit-for-tat strategy to cope with free-riding in unreliable p2p networks. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, Washington, DC, USA, pp. 336–341. IEEE Computer Society.

Ratnasamy, S., P. Francis, M. Handley, R. Karp, and S. Shenker (2001). A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pp. 161–172.

Resnick, P., R. Zeckhauser, J. Swanson, and K. Lockwood (2003). The value of reputation on ebay: A controlled experiment. *Experimental Economics 9*, 79–101.

Rowstron, A. and P. Druschel (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware*, 329–350.

Saroiu, S., P. K. Gummadi, and S. D. Gribble (2002). A measurement study of peer-to-peer file sharing systems.

Sherman, C. S. (2005). Running on karma- p2p reputation and currency systems. In *Proceedings of the 6th international conference on Cryptology and network security*.

Srivatsa, M., L. Xiong, and L. Liu (2005). Trustguard: Countering vulnerabilities in reputation management for decentrailized overlay networks.

Stoica, I., D. Adkins, S. Zhuang, S. Shenker, and S. Surana (2002). Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM*, pp. 73–86.

Tamilmani, K., V. Pai, and A. E. Mohr (2004). Swift: A system with incentives for trading.

Vanhoose, D. D. (2011). Multiple currency economies.

Vishnumurthy, V., S. Chandrakumar, S. Ch, and E. G. Sirer (2003). Karma: A secure economic framework for peer-to-peer resource sharing.

Xiong, L. and L. Liu (2004). Peertrust: Supporting reputation-based trust in peer-to-peer communities. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

Zynga (2011). Zynga - connecting the world through games. http://www.zynga.com/gamecards/info/.