Resilient Reputation and Trust Management:
Models and Techniques

A Thesis
Presented to
The Academic Faculty

by

Li Xiong

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005
Resilient Reputation and Trust Management:
Models and Techniques

Approved by:

Dr. Ling Liu, Advisor
College of Computing
Georgia Institute of Technology

Dr. Calton Pu
College of Computing
Georgia Institute of Technology

Dr. Mustaque Ahamad
College of Computing
Georgia Institute of Technology

Dr. Munindar Singh
Department of Computer Science
North Carolina State University

Dr. Leo Mark
College of Computing
Georgia Institute of Technology

Dr. D.J. Wu
College of Management
Georgia Institute of Technology

Date Approved: August 26th, 2005
To my beloved family
ACKNOWLEDGEMENTS

I am indebted to many people for their guidance and support along the path to this dissertation.

First, I would like to thank my thesis advisor Professor Ling Liu. I would not have come this far this quickly without her unfailing encouragement and guidance. She has taught me much about research, the academic world, and life. Her keen attention to details and vision for the big picture were indispensable in shaping this dissertation. She is always encouraging and caring. She always makes time and efforts when I need her advice. I hope that I can be as great an advisor to my students as she has been to me.

My sincere thanks go to my thesis committee members Professor Mustaque Ahamad, Leo Mark, Calton Pu, Munindar Singh, and D.J. Wu. Their invaluable comments have helped me refine my thesis in a much better way. They have also shared many of their experiences and perspectives with me on many aspects of the academic world and have never failed to give me good advice.

Being affiliated with the Distributed Data Intensive Systems Lab and the Center for Experimental Research in Computer Systems and working with many wonderful people has made my time at Georgia Tech enjoyable. Parts of this dissertation research are results of collaboration with Mudharkar Srivatsa and Subburamnyam Chitti. It was a rewarding experience. James Caverlee and Steve Webb are great officemates. We had wonderful conversations and shared many laughs as well as frustrations. Lakshmi Ramaswamy and I shared many experiences and helped each other as we are finishing up about the same time. Thanks to all the people in the lab for the many discussions, practice talks, lunches and everything.

I can not thank my family enough for their love and support. My parents have always encouraged my education, respected my own choices at each turning point of my life, and given me endless support along the way. For this and much more, I am forever indebted
to them. My parents-in-law have offered tremendous help and support. I am sincerely grateful.

Finally, I owe my thanks and love to my husband Wei for all the wonderful years we have been together. Through the ups and downs of my research, he is always there offering help, support, encouragement, and love. We welcomed our son Brandon during this dissertation period. It was hard to juggle between parenting responsibilities and research at times, but more than anything, it has been a joy and an amazing experience. As I embark on a new journey, I look forward to many new beginnings in my life with my family by my side.
TABLE OF CONTENTS

DEDICATION ............................................................... iii

ACKNOWLEDGEMENTS ................................................... iv

LIST OF TABLES .......................................................... x

LIST OF FIGURES .......................................................... xi

SUMMARY ................................................................. xiii

I  INTRODUCTION ......................................................... 1
  1.1 Application and Risk Scenarios ................................ 1
  1.2 Reputation Systems and Research Challenges ............... 4
  1.3 Assumptions and Problem Scope .............................. 6
  1.4 PeerTrust: Contributions and Impacts ...................... 9
  1.5 Outline .......................................................... 12

II  BASIC TRUST FRAMEWORK ......................................... 13
  2.1 Problem Statement ............................................... 13
  2.2 The Trust Model ................................................ 14
    2.2.1 Trust Parameters ......................................... 15
    2.2.2 General Trust Metric ..................................... 17
    2.2.3 The Basic Metric .......................................... 18
    2.2.4 Adapting the Trust Metric with Context Factors ...... 20
  2.3 Implementation Strategies .................................... 21
    2.3.1 Managing Trust Data: System Architecture ............ 22
    2.3.2 Trust Computation ....................................... 24
    2.3.3 Dealing with Dynamic Personality of Peers .......... 27
    2.3.4 Secure Processing and Transmission of Trust Data .... 29
    2.3.5 Trust-based Peer Selection Scheme ..................... 30
  2.4 Experimental Evaluation ....................................... 31
    2.4.1 Experiment Setup ......................................... 31
    2.4.2 Effectiveness against Malicious Behaviors of Peers .... 33
    2.4.3 Benefit of the Trust Based Peer Selection ............ 36

vi
2.4.4 Effectiveness against Dynamic Personality of Peers .......... 37
2.4.5 Trust Evaluation Cost ................................................. 39
2.5 Summary ................................................................. 40

III COUNTERING FEEDBACK SPARSITY ................................. 42
3.1 Introduction ............................................................... 42
3.2 Problem Statement ....................................................... 45
3.2.1 Terms and Definitions .............................................. 45
3.2.2 Threat Model ............................................................ 47
3.2.3 Sparsity Problem ....................................................... 49
3.3 A Similarity Inference Scheme ....................................... 50
3.3.1 Overview ............................................................... 51
3.3.2 Computing Credibility using Similarity ......................... 53
3.3.3 Similarity Inference .................................................. 55
3.4 Experimental Evaluations ............................................. 61
3.4.1 Experiment Setup ..................................................... 61
3.4.2 Evaluation Metrics ................................................... 62
3.4.3 Countering Sparsity .................................................. 63
3.4.4 Effect of Feedback Manipulation ................................. 65
3.5 Summary ................................................................. 71

IV COUNTERING FEEDBACK OSCILLATION ............................. 72
4.1 Threat Model and Design Goals ..................................... 73
4.2 Dependable Trust Model ................................................ 74
4.2.1 Trust Computation Model ......................................... 75
4.2.2 Realizing the Trust Model ......................................... 76
4.2.3 Optimization through Fading Memories ......................... 77
4.3 Experimental Evaluation ............................................... 79
4.3.1 Experiment Setup ..................................................... 80
4.3.2 Comparing Optimistic and Pessimistic Summarizations ....... 80
4.3.3 Trust Model Parameters ............................................ 81
4.3.4 Varying History Size ................................................. 82

vii
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.5</td>
<td>Fading Memories</td>
<td>83</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Other Strategic Oscillation Models</td>
<td>84</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>84</td>
</tr>
<tr>
<td>V</td>
<td>PRESERVING FEEDBACK PRIVACY: PRIMITIVE PROTOCOLS</td>
<td>86</td>
</tr>
<tr>
<td>5.1</td>
<td>Problem Statement</td>
<td>87</td>
</tr>
<tr>
<td>5.2</td>
<td>Problem Statement</td>
<td>90</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Adversary Model</td>
<td>90</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Privacy Goal</td>
<td>91</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Privacy Metrics</td>
<td>92</td>
</tr>
<tr>
<td>5.3</td>
<td>The Decentralized Protocol</td>
<td>94</td>
</tr>
<tr>
<td>5.3.1</td>
<td>A Naive Protocol</td>
<td>95</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Protocol Structure</td>
<td>95</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Privacy Preserving Max Selection</td>
<td>96</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Privacy Preserving Top-k Selection</td>
<td>100</td>
</tr>
<tr>
<td>5.4</td>
<td>Analysis</td>
<td>103</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Correctness</td>
<td>103</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Efficiency</td>
<td>104</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Data Value Privacy</td>
<td>105</td>
</tr>
<tr>
<td>5.5</td>
<td>Experimental Evaluation</td>
<td>108</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Experiment Setup</td>
<td>109</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Precision of Max Selection</td>
<td>109</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Privacy Characteristics of Max Selection</td>
<td>110</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Precision of Top-k Selection</td>
<td>113</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Privacy Characteristics of Top-k Selection</td>
<td>114</td>
</tr>
<tr>
<td>5.6</td>
<td>Summary</td>
<td>115</td>
</tr>
<tr>
<td>VI</td>
<td>PRESERVING FEEDBACK PRIVACY: AGGREGATION</td>
<td>116</td>
</tr>
<tr>
<td>6.1</td>
<td>Problem Statement</td>
<td>116</td>
</tr>
<tr>
<td>6.2</td>
<td>Design Issues</td>
<td>120</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Problem Definition and Threat Model</td>
<td>120</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Evaluation Metrics</td>
<td>120</td>
</tr>
</tbody>
</table>
6.3 Privacy Preserving \textit{k}NN Classification Model .................. 122
  6.3.1 Classification Problem and the \textit{k}NN Classifier .................. 122
  6.3.2 Privacy Preserving \textit{k}NN Classification ...................... 123
  6.3.3 Analysis of the Model ........................................ 125
6.4 The Algorithm .................................................. 126
  6.4.1 Privacy Preserving Nearest Neighbor Selection .................. 126
  6.4.2 Privacy Preserving Classification ............................. 127
  6.4.3 Privacy Preserving \textit{k}NN Classification ...................... 129
6.5 Analysis ........................................................... 129
  6.5.1 Accuracy ..................................................... 130
  6.5.2 Efficiency ................................................... 130
  6.5.3 Privacy ....................................................... 131
6.6 Experimental Evaluation ........................................... 133
  6.6.1 Experiment Setup ............................................. 133
  6.6.2 Relative Accuracy ........................................... 134
  6.6.3 Effect of Number of Rounds .................................. 135
  6.6.4 Privacy Characteristics ..................................... 136
6.7 Summary .......................................................... 137

VII RELATED WORK .................................................. 138
  7.1 Trust Management in Distributed Systems Security .................. 138
  7.2 Reputation and Trust in Electronic Communities .................... 140
    7.2.1 Reputation and Trust in E-Commerce and Agent Communities ... 141
    7.2.2 Reputation Systems for P2P Networks .......................... 143
  7.3 Recommender Systems ............................................. 146
  7.4 Privacy Preserving Data Management ................................ 149

VII CONCLUSION AND OPEN ISSUES .................................. 151

REFERENCES ......................................................... 154

VITA ................................................................. 161
# LIST OF TABLES

<table>
<thead>
<tr>
<th></th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic Trust Framework: Experiment Parameters</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>Countering Feedback Sparsity: Experiment Parameters</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>Countering Feedback Sparsity: Attack Models</td>
<td>62</td>
</tr>
<tr>
<td>4</td>
<td>Primitive Protocol: Experiment Parameters</td>
<td>109</td>
</tr>
<tr>
<td>5</td>
<td>$k$NN Classification: Experiment Parameters</td>
<td>134</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PeerTrust System Architecture</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Trust Computation Error with Varying Percentage of Malicious Peers</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>Trust Computation Error with Varying Malicious Rate of Malicious Peers</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>The Benefit of Trust Based Peer Selection Scheme</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Effectiveness against Dynamic Personality and Reputation Oscillation</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Trust Computation Overhead</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>Similarity Inference Scheme: Illustration</td>
<td>51</td>
</tr>
<tr>
<td>8</td>
<td>Discovering Indirect Neighbors: Illustration</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>Trust Computation and Coverage of Different Similarity Measure with Varying Sparsity Levels</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>Trust Computation Error and Coverage of Varying Maximum Path Length with Varying Sparsity Levels</td>
<td>64</td>
</tr>
<tr>
<td>11</td>
<td>Trust Computation Error and Coverage of Different Path Combination Functions with Varying Sparsity Levels</td>
<td>65</td>
</tr>
<tr>
<td>12</td>
<td>Trust Computation Error and Coverage of Varying Neighbor Selection Threshold with Varying Sparsity Levels</td>
<td>65</td>
</tr>
<tr>
<td>13</td>
<td>Trust Computation Error and Coverage of Different Similarity Measures in Non-Collusive Model</td>
<td>66</td>
</tr>
<tr>
<td>14</td>
<td>Trust Computation Error and Coverage of Different Similarity Measures in Collusive Model</td>
<td>66</td>
</tr>
<tr>
<td>15</td>
<td>Trust Computation Error and Coverage of Different Similarity Measures in Collusive Camouflage Model</td>
<td>67</td>
</tr>
<tr>
<td>16</td>
<td>Trust Computation Error and Coverage of Varying Maximum Path Length in Non-Collusive Model</td>
<td>67</td>
</tr>
<tr>
<td>17</td>
<td>Trust Computation Error and Coverage of Varying Maximum Path Length in Collusive Model</td>
<td>68</td>
</tr>
<tr>
<td>18</td>
<td>Trust Computation Error and Coverage of Varying Maximum Path Length in Collusive Camouflage Model</td>
<td>68</td>
</tr>
<tr>
<td>19</td>
<td>Trust Computation Error and Coverage of Different Path Combination Functions in Non-Collusive Model</td>
<td>69</td>
</tr>
<tr>
<td>20</td>
<td>Trust Computation Error and Coverage of Different Path Combination Functions in Collusive Model</td>
<td>69</td>
</tr>
</tbody>
</table>
21  Trust Computation Error and Coverage of Different Path Combination Functions in Collusive Camouflage Model ......................................................... 70
22  Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Non-Collusive Model ............................................................ 70
23  Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Collusive Model ................................................................. 71
24  Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Collusive Camouflage Model .................................................... 71
25  Cost of Building Reputation .................................................................................................................................................................................. 74
26  PID Based Dependable Trust Model: Illustration ........................................................................................................................................... 75
27  Updating Fading Memories: Illustration ...................................................................................................................................................... 78
28  Oscillating Behavior Model I ................................................................................................................................................................................ 80
29  Optimistic Vs, Pessimistic Summarization .................................................................................................................................................... 81
30  Effect of Varying Parameters in the PID Trust Model ................................................................................................................................. 81
31  PID Trust Model with Varying History Size ................................................................................................................................................ 82
32  PID Trust Model with Fading Memories ....................................................................................................................................................... 83
33  Privacy Preserving Max Protocol: Illustration .............................................................................................................................................. 99
34  Local Algorithm of Privacy Preserving Topk Protocol: Illustration .............................................................................................................. 102
35  Precision Guarantee with Number of Rounds ........................................................................................................................................... 104
36  Required Number of Rounds with Precision Guarantee ............................................................................................................................... 105
37  Expected Loss of Privacy in Different Rounds .......................................................................................................................................... 107
38  Precision of Max Selection with Number of Rounds ..................................................................................................................................... 109
39  Loss of Privacy for Max Selection in Different Rounds ............................................................................................................................... 110
40  Loss of Privacy for Max Selection with Different Number of Nodes ........................................................................................................... 111
41  Tradeoff between Privacy and Efficiency with Varying Randomization Parameters ................................................................................................ 112
42  Comparison of Loss of Privacy with Number of Nodes ............................................................................................................................... 113
43  Precision of Topk Selection with Number of Rounds ..................................................................................................................................... 114
44  Comparison of Loss of Privacy with \( k \) .............................................................................................................................................................. 114
45  Accuracy Comparison of Privacy Preserving Classifier and Regular Classifier ............................................................................................ 134
46  Relative Accuracy with Varying Number of Rounds in Nearest Neighbor Selection .......................................................................................... 135
47  Loss of Privacy in Nearest Neighbor Selection .............................................................................................................................................. 136
The continued advances in distributed service-oriented computing and global communications have created a strong technology push for online information sharing and business transactions among enterprises, organizations, and individuals. While these communities offer enormous opportunities, they also present potential threats and risks due to a lack of trust. Reputation systems provide a way for building trust through social control by harnessing the community knowledge in the form of feedback. Although feedback-based reputation systems help community participants decide who to trust and encourage trustworthy behavior, they also introduce vulnerabilities due to potential manipulations by dishonest or malicious players. Therefore, building an effective and resilient reputation system remains a big challenge for the wide deployment of service-oriented computing.

This dissertation proposes a decentralized and dependable reputation based trust supporting framework called PeerTrust, focusing on models and techniques for resilient reputation management against feedback aggregation related vulnerabilities, especially feedback sparsity with potential feedback manipulation, feedback oscillation, and loss of feedback privacy. This dissertation research has made three unique contributions for building a resilient decentralized reputation system.

The first main contribution is the development of a basic reputation-based trust framework that helps establishing trust among participating parties based on their past experiences (such as transaction histories and feedback) in a P2P community. We identify important trust parameters by analyzing the common problems of building a reputation system. We provide a coherent trust metric for quantifying and comparing the trustworthiness and predicting the future behavior of individual participants, using transaction-based feedbacks. We also propose strategies for implementing the trust model in a decentralized, efficient and secure manner. Our experimental evaluation shows the feasibility, effectiveness,
cost and benefit of this reputation model and its trust evaluation framework.

The second contribution is the refinement of the trust framework to minimize the potential threats and vulnerabilities in the reputation system itself. We develop threat models and defense mechanisms that enable the framework to be resilient to several vulnerabilities, including the risks involved in the presence of sparse feedbacks with potential feedback manipulations, and strategic oscillating behaviors of some participants. First, we develop a similarity inference framework to counter feedback sparsity with potential feedback manipulations. We propose similarity measures for differentiating dishonest feedbacks from honest ones and propose an inference framework to address the sparsity issues. We perform extensive evaluations of various algorithmic component of the framework and evaluate their effectiveness in countering feedback sparsity. Second, we introduce a Proportional, Integral, and Derivative (PID) based model to effectively handle strategic oscillating behavior of malicious nodes. The model promotes the incorporation of the reputation history and behavior fluctuations of nodes into the estimation of their trustworthiness. Our experimental evaluation shows the system is highly dependable against various strategic oscillating behaviors of peers.

The third main contribution addresses the problem of loss of privacy in the reputation system and the challenge of sharing data while respecting privacy constraints of the individual parties. We first develop a set of novel probabilistic privacy-conscious computation protocols for important primitive operations among multiple private databases or individuals (such as max, min, and topk). As a tradeoff for efficiency and practicability, the constraint of not revealing any additional information apart from the final result is relaxed to allow minimal additional information to be revealed. Our analysis and experimental evaluations show that our protocols on one hand effectively minimize the information disclosure of individual users and on the other hand are efficient in terms of both computation and communication costs. We next show how feedback aggregation can be divided into individual steps that utilize above primitive protocols. In particular, we model a set of reputation computation algorithms as kNN classification problem and propose a model for privacy preserving kNN classification.
CHAPTER I

INTRODUCTION

The continued advances in distributed service-oriented computing and global communications have created a strong technology push for online information sharing and business transactions among enterprises, organizations, and individuals. These participating parties are increasingly forming online communities to communicate directly with one another to exchange information, distribute tasks, or execute transactions. While these communities offer enormous opportunities, they also present potential threats and risks. We start the dissertation by motivating our problem and discussing application and risk scenarios in these communities.

1.1 Application and Risk Scenarios

There are several driving forces behind the trend for online information sharing and business transactions,

- Business-to-Business Services. In the business world, with the push of end-to-end integration between enterprises and their suppliers, service providers, and trading partners, organizations are collaborating with constantly evolving alliances. Business transactions may occur across multiple autonomous enterprises which are actively competing as well as collaborating. In federal and local governments, agencies realize the importance of sharing information for devising effective security measures and for cooperating in response to massively disruptive events. Multiple agencies may need to share their criminal record databases for identifying certain suspects under the circumstance of a terrorist attack while minimizing data disclosure of each individual agency.

- Person-to-Person Marketplaces. Online marketplaces are becoming increasingly popular by providing a global trading platform for the sale of goods and services by a
diverse community of individuals and small businesses. For example, the eBay community includes more than a hundred million registered members. There are millions of items listed everyday and people from all over the world buy and sell items in thousands of categories.

- **Peer-to-Peer Networks.** Despite its legal controversy, P2P networks are increasingly gaining acceptance on the Internet as they provide an infrastructure in which the desired information can be located and traded while preserving the anonymity of both requestor peers and provider peers. Peers join the community sharing a variety of content files containing audio, video, and other data in digital format. There are also networks such as FreeNet that harness the computing power of individual peers to achieve a global computation task that are otherwise unachievable.

These communities can be implemented either on top of a P2P network or using a conventional client-server platform. Pure P2P networks such as Gnutella are examples that are built on top of a P2P platform. Person-to-person online auction sites such as eBay and many business-to-business (B2B) services such as supply-chain-management networks are examples that are built on top of client-server architecture.

While these communities offer enormous opportunities, they also present potential threats and risks due to a *lack of trust.* The communities are often established dynamically with interested parties who may be unrelated and unknown to each other. Further, they are anonymous and are only identified by a pseudo-identifier. For example, Gnutella servants (peers) are only identified by a self-identified servant ID. eBay users trades with a self-identified user ID. This open and anonymous nature leads to a lack of trust and accountability and opens the door to possible misuses and abuses by malicious peers. As a result, organizations and individuals have to deal with risks that are associated with potentially untrustworthy parties and have to cope with much higher amount of uncertainty as to quality and reliability of the products they buy and the information they obtain from others in the respective communities. Further, these communities provide the infrastructure of trading a variety of information, products, and services. For instance, the files authorized
to be shared in Gnutella can include all media types, including executable and binary files. This feature combined with the lack of trust make the P2P environments more vulnerable to certain security attacks such as distribution of tampered with information.

Below we summarize a number of security threats and risks in such open distributed communities.

- **Business Transaction Risks.** In communities that trade products, peers have to cope with much higher amount of uncertainty as to quality and reliability of the products they buy. For example, in online auction communities, buyers are vulnerable to potential risks because malicious sellers may provide incomplete or distorted information of the products they are trying to sell or even fail to deliver the products after the payment. It is important to help the community build trust with one another and keep the marketplace a safe place to trade.

- **Distribution of Tampered-with Information.** As recent experiences with Gnutella show, peers can exploit the overlay network as a way to spread tampered-with information, including malicious programs, such as Trojan Horses and viruses. For example, when a user is requesting a resource from the network, an adversary can respond with a fake resource with the same name as the real resource the original user is looking for. The actual file however could be a Trojan horse program or a virus like the well-known VBS.Gnutella worm. Similarly, in a mobile software downloading scenario where a mobile user is asking for a resource from the network, an adversary can also distribute a wireless virus such as PalmOS/Phage that has been discovered in PalmOS and will infect all third party applications on the PDA device.

- **Denial of Service Attacks.** Peers can send unsolicited information across the network and make the network run very slowly or break down completely. For example, the first cell phone virus hacked users of GSM mobile phones and broadcasted a disparaging remark through Short Messaging Service (SMS). Although the virus caused no damage, it foreshadowed a potential Denial of Service (DoS) attack. If an adversary can disseminate a worm that will send out millions of such messages, it could deluge
cell phones with them, thereby overwhelming the messaging system.

- **Data Disclosure.** Another risk associated with sharing information among communities is the potential disclosure of private or secret data by untrustworthy or malicious partners. For example, when multiple agencies share their criminal record data and indiscriminately open up their databases to each other, the information may be disclosed to public by a malicious participant. Thus selective file sharing among trustworthy partners is extremely important in such mission critical applications.

Given these potential risks and threats, it is important to develop tools and mechanisms that minimize the risks and threats and help make these communities a safe place to trade. Conventional security techniques such as cryptography can address or alleviate some of the risks but cannot help peers to establish trust with each other. We need to develop strategies for establishing trust and develop systems that can assist peers in assessing the level of trust they should place on each other and avoid malicious parties. For example, in the buyer-seller market, trust is critical as it can provide buyers with high expectations of satisfying exchange relationships.

### 1.2 Reputation Systems and Research Challenges

Recognizing the importance of trust in such communities, an immediate question to ask is how to establish trust. There has been a large amount of research [84] focusing on formal models of trust propagation based on digital credentials in the name-key binding problem in computer security area. However, digital credentials based approach tends to be expensive and it can not be always enforced, especially in the open communities that we are considering. There is also extensive amount of research focused on building trust for electronic markets through trusted third parties or intermediaries [52, 11]. However, finding such a trusted third party is not always feasible. The level of trust required for the third party with respect to intent and competence against security breaches is too high. It is also widely recognized that a trusted third party based solution is not scalable, has a single point of failure, and incurs heavy administrative overheads.
Reputation systems [73, 106, 98, 27] provide a way for building trust through social control without relying on trusted third parties. Concretely, they help participants to evaluate trustworthiness of each other and predict future behaviors of participants based on the community feedback about the participants’ past behavior. By harnessing the community knowledge in the form of feedback, these systems help people decide who to trust, encourage trustworthy behavior, and deter dishonest participation. Reputation systems are important for fostering trust and minimize risks in two ways. First, by collecting and aggregating feedback about participants’ past behavior, they provide a way for participants to share their experiences and knowledge so they can estimate the trustworthiness of other participants with whom they may not have personal experiences and in turn they can avoid malicious participants to reduce risk. Second, the presence of a reputation system creates the expectation of reciprocity or retaliation in future behavior, which in turn creates an incentive for good behavior and discourages malicious behavior.

A variety of online community sites have reputation management built in, such as eBay, Amazon, Yahoo!Auction, Edel, Slashdot, and Entrepreneur.com. Even though these sites facilitate the trust among users to some extent, they also have some common problems and vulnerabilities. Most of these systems use a simple sum or average of the ratings as the reputation value of a user. For example, eBay uses a summation of positive and negative feedback. It fails to convey important subtleties of online interactions such as whether these feedback ratings come from low-value transactions and whether the feedback ratings are honest. It is important to develop effective metrics that aggregate feedback into a meaningful trust value as an estimate of the trustworthiness of participants by incorporating all the subtleties of online interactions.

In addition, a reputation system also introduces vulnerabilities itself. Malicious users can exploit or manipulate the system for their own benefit. There are a number of attacks that are identified for reputation systems and we list them below.

- **Shilling attacks.** Malicious users can have or create multiple users (shills) that provide misleading feedbacks in order to boost their own ratings or damage the ratings of other peers.
• **Whitewashing attacks.** If a malicious peer finds its reputation has dropped due to its malicious behavior, it can abandon its old identity and reenter the network with a new identity to get rid of its bad history.

• **Traitor attacks.** A malicious node may behave non-maliciously until it attains a good reputation (reflected in its trust value) and then behave maliciously.

Thus the design goal of a reputation system is that it should be not only accurate and effective in estimating the trustworthiness of a user by incorporating all the subtleties of online interactions but also resilient to various malicious behaviors of users such as the above attacks.

### 1.3 Assumptions and Problem Scope

There are many interesting issues to be explored in order to develop a resilient reputation trust management system. Many of these issues span multiple areas including information science, electronic commerce, agent systems, and computer security. In this section, we first define our problem scope by introducing a few assumptions we make throughout the dissertation and summarizing the main research challenges we address in developing resilient reputation and trust management.

First, we consider an open community that consists of potentially a large number of malicious nodes. The game theory research on reputation introduced two types of players [27]. One is commitment type or a long-run player who would always cooperate because cooperation is the action that maximizes the player’s lifetime payoffs if the player could credibly commit to an action for the entire duration. In contrast, a strategic or malicious type corresponds to an opportunistic player who cheats whenever it is advantageous for him to do. This dissertation focuses on developing models and techniques that are resilient to such potential malicious behaviors.

Second, although malicious peers can behave in any arbitrary ways, we focus on a few vulnerabilities in this dissertation and assume certain behavior models of malicious peers. We aim to model peers’ malicious behavior realistically by analyzing the attack intent and
strategies of malicious peers and we define threat models within each chapter for a given
vulnerability we consider. Given a threat model, our objective is to maximize the cost
(penalty) to be paid by the malicious nodes in terms of contributing to the community by
providing good service and honest feedback.

Finally, we assume that the reputation system architecture is built on top of a secure
overlay network and each node is identified by a pseudo-identifier. Thus, the overlay network
should be capable of routing messages despite the presence of some malicious nodes and
ensure that all nodes can be identified through some digital certification based mechanism.
Readers may refer to [56, 29, 81] for a detailed discussion on security issues in overlay
networks.

Bearing these assumptions in mind, we focus on a few main research challenges in
building a resilient reputation system in this dissertation.

- **Inaccurate or dishonest feedback.** Given the main task of aggregating the user feed-
  back, the main challenge of a reputation system is how to effectively cope with the
  quality of user feedback. This includes erroneous or inaccurate feedback by normal
  users as well as dishonest feedback by malicious users. Malicious users may provide
  false or misleading feedback to boost their own rating, to badmouth other participants,
  or just to fool the system. Things are made much worse if a group of malicious par-
  ticipants collude to boost the ratings of each other and damage the ratings of others.
  Shilling attacks are one such example of achieving malicious motives through dishonest
  feedbacks. An effective trust metric has to differentiate inaccurate or dishonest
  feedback from honest ones and be robust against various malicious manipulations of
  participants.

- **Sparse feedback.** Related to the quality of the feedback, there is also a lack of feedback
due to the sheer size of certain communities and a lack of incentive for participants
to provide feedback. The lack of incentive to provide feedback could be a problem in
particular in certain communities such as mobile commerce where mobile users may
not bother to provide feedback due to the power limitations of their mobile devices
and their on-the-road situation. It is difficult for the reputation system to operate on an extreme sparse feedback data.

- **Context awareness.** Another important consideration is the trust context, as many of the applications are sensitive to the context of the transactions. For example, the functionality of the transaction is an important context to be incorporated into the trust metric. Amazon.com may be trustworthy on selling books but not on providing medical devices. Most systems provide no support to incorporate various contexts in evaluating the trustworthiness of peers. For example, a peer can develop a good reputation by being honest for numerous small transactions and then try to make a profit by cheating for large transactions.

- **Dynamic behavior.** More general to the traitor attack, one of the detrimental vulnerabilities is that a malicious node may strategically alter its behavior in a way that benefits itself. Most existing reputation systems such as eBay use a combination of average feedbacks and the number of transactions performed by a node as indicators of its trust value. As a result, a bad node can perform traitor attack by behaving non-maliciously until it attains a good reputation and then behave maliciously. It could also oscillate between building and milking reputation. A highly dependable reputation system is needed to safeguard the system itself from these vulnerabilities and should be able to penalize malicious nodes for such dynamic and strategic behavioral changes.

- **Loss of Privacy.** Lastly, reputation aggregation also raises privacy concerns as participants may not wish to share their feedback with others for various reasons. For example, a user who has a low rating about a malicious user may not be willing to share its rating as the malicious user may track down the users who give him low ratings and harass or legally attack them. A research challenge is how to share the reputation data while respecting the privacy constraints of the data owners.

Interestingly, this is also a general risk of sharing data in open communities (recall the risks that we listed for open communities). Because of the open and untrusted
environments, individual peers or databases are not willing to fully disclose their private data. Concretely, given an information integration task spanning multiple private databases or individuals, we wish to compute the answer without revealing any information on each individual database apart from the query result.

In addition to the above challenges we have outlined for building an effective trust model, the effectiveness of a trust system also depends on the implementation of the trust model in a decentralized system. The important challenge is how to build the supporting infrastructure to collect, aggregate and distribute feedback and reputation information.

- **Efficient and scalable reputation data dissemination.** There are two alternative ways for reputation data dissemination, namely centralized and decentralized. A trust model can be implemented by either scheme. For example, in the mobile commerce communities that are built on top of client-server architecture, a centralized trust server (service provider or other independent service provider) can be deployed to collect, aggregate and distribute reputation information.

  The nature of P2P networks makes the traditional centralized solution unfeasible as there is no centralized server or database. It remains a challenge to build a decentralized trust management system that is efficient and scalable in both trust computation and trust data storage and dissemination.

- **Secure Trust Data Transmission.** Another important issue with the management of the feedback and trust data is that the reputation system infrastructure has to guarantee the secrecy and integrity of the reputation data during their transmission.

### 1.4 PeerTrust: Contributions and Impacts

Focusing on the research challenges we have presented, this dissertation proposes a decentralized and dependable reputation based trust supporting framework that helps establishing trust among participating parties. We first develop a basic trust framework including a trust model with important trust parameters identified and a set of decentralized implementation strategies. We then refine the framework and develop models and techniques to
make the framework more resilient and dependable. In particular, we focus on feedback aggregation related vulnerabilities, including feedback sparsity, feedback oscillation, and loss of feedback privacy, and develop countermeasures against them. Below we summarize the main contributions of this dissertation.

**Basic Trust Framework.** The first main contribution is the development of a basic reputation-based trust framework that helps establishing trust among participating parties based on their past experiences (such as transaction histories and feedback) in a P2P community. We identify important trust parameters by analyzing the common problems of building a reputation system. We provide a coherent trust metric for quantifying and comparing the trustworthiness and predicting the future behavior of individual participants, using transaction-based feedbacks.

We also propose strategies for implementing the trust model in a decentralized, efficient and secure manner. Our initial experimental evaluation shows the feasibility, effectiveness, cost and benefit of this reputation model and its trust evaluation framework.

**Countering Feedback Vulnerabilities.** The second contribution of this dissertation is the refinement of the trust framework to minimize the potential threats and vulnerabilities in the reputation system itself. We develop threat models and defense mechanisms that enable the framework to be resilient to several vulnerabilities, including dishonest feedbacks, the risks involved in the presence of sparse feedbacks, and strategic and malicious behaviors of some participants.

- **Countering Feedback Sparsity.** First, we develop a similarity inference framework to counter feedback vulnerabilities that are incurred due to dishonest feedbacks and sparse feedbacks. We adapt and extend existing collaborative filtering techniques and propose similarity measures for differentiating dishonest feedbacks from honest ones and propose an inference framework to address the sparsity issues. We perform extensive evaluations of various algorithmic component of the framework and evaluate their effectiveness in countering the feedback vulnerabilities.

- **Countering Feedback Oscillation.** Second, we introduce a Proportional, Integral, and
Derivative (PID) based model to effectively handle strategic oscillating behaviors of malicious nodes. We promote the incorporation of the reputation history and behavior fluctuations of nodes into the estimation of their trustworthiness. We use adaptive parameters to allow different weighting functions to be applied to current reputation, reputation history, and reputation fluctuations. Our experimental evaluation shows the system is highly dependable against various strategic oscillating behaviors of peers.

Preserving Feedback Privacy The third main contribution addresses the problem of loss of privacy in the reputation system and the challenge of sharing data while respecting privacy constraints of the individual parties. We propose a model in which users control their feedback data and propose protocols whereby a set of users can compute a public aggregation of their feedback data that does not expose individual users. The aggregation essentially allows users to compute reputation values of other users based on certain trust metrics.

- **Primitive Privacy Preserving Protocols** We tackle the problem by first developing a set of novel decentralized privacy-conscious computation protocols for important primitive operations among multiple private databases or individuals (such as MAX, MIN, and TopK). We adopt the paradigm of information integration with minimal necessary sharing [9] for developing privacy-preserving computation protocols. As a tradeoff for efficiency and practicability, the constraint of not revealing any additional information apart from the final result is relaxed to allow minimal additional information to be revealed. Our analysis and experimental evaluations show that our protocols on one hand effectively minimize the information disclosure of individual users and on the other hand are efficient in terms of both computation and communication costs.

- **Privacy Preserving Feedback Aggregation.** We next show how feedback aggregation can be reduced or divided into individual steps that utilize above primitive protocols of individual functions. In particular, we model a set of reputation computation algorithms as kNN classification problem and propose a model for privacy preserving kNN classification. We illustrate how the model can be implemented with the topk
protocol proposed above and an existing summation protocol.

It is worth mentioning that these contributions are not limited to building a privacy-conscious reputation trust framework but also apply to the general problem of preserving privacy for information integration across private databases.

In summary, the PeerTrust framework presents an important step forward with respect to developing attack-resilient reputation trust systems that are important for the wide deployment of service-oriented computing. Of course, there are many remaining questions with the risks that we studied and the approach we proposed. There are also other risks and vulnerabilities that are not directly addressed in this dissertation. We will discuss them as open issues in Chapter 8.

1.5 Outline

The rest of the dissertation is organized as follows. Chapter 2 presents the basic reputation-based trust framework. Chapter 3 presents the similarity inference framework for countering feedback sparsity and dishonest feedback. Chapter 4 presents the models and techniques we developed in dealing with strategic oscillating behaviors of nodes. Chapter 5 presents a set of primitive decentralized privacy preserving protocols for information aggregation operations such as max, min, and topk. Chapter 6 models a set of reputation algorithms as kNN classification and presents a privacy preserving kNN classification framework built on top of the primitive protocols. While some related work is discussed within each chapter as appropriate, Chapter 7 provides an overview of the research in reputation and trust related area. Finally, Chapter 8 concludes the dissertation and discusses future work and open issues.

Parts of the dissertation have been published in conferences and journals. In particular, the basic PeerTrust model and framework is described in a CEC-2003 paper [94] and a subsequent journal version in TKDE 2004 [95]. The model for dealing with strategic and dynamic behaviors of malicious peers is described in a WWW-2005 [82] paper. The key ideas and protocols for computing primitive functions across private databases are described in an ICDCS-2005 [93] paper.
CHAPTER II

BASIC TRUST FRAMEWORK

We first present a basic P2P reputation-based trust supporting framework. The framework has a number of unique contributions. First, by analyzing a variety of common problems encountered in today’s online communities (Section 2.1), we introduce the trust model (Section 2.2) with five important parameters and a general trust metric combining these parameters for evaluating the trustworthiness of a peer in an evolving P2P community. We also present the trust information dissemination architecture, the usage of the trust model, the design and implementation considerations of the framework (Section 2.3). Finally we describe a series of simulation-based experiments that are carefully designed to evaluate the framework by showing its accuracy, robustness, cost, and efficiency (Section 2.4). We conclude the chapter with a brief summary (Section 2.5).

2.1 Problem Statement

P2P electronic communities are increasingly gaining acceptance on the Internet as they provide an infrastructure in which the desired information and products can be located and traded while preserving the anonymity of both requestor peers and provider peers. As recent experience with P2P systems such as Gnutella shows, anonymity opens the door to possible misuses and abuses by malicious peers exploiting the overlay network as a way to spread tampered with information, including malicious programs, such as Trojan Horses and viruses. One way to minimize threats in an open community as such is to use community-based reputations, which can be computed through feedback about peers’ transaction histories.

A variety of online community sites have reputation management built in, such as eBay, Amazon, Yahoo!Auction, Edeal, Slashdot, Entrepreneur.com.
From our experience with these sites, and the survey provided in [57, 73, 27], we summarize a list of common problems and risks observed in the current P2P e-commerce communities.

- Most existing reputation systems lack ability to differentiate dishonest feedback from honest ones and hence are vulnerable to malicious manipulations of peers who provide dishonest feedback.
- Most systems provide no support to incorporate various contexts in evaluating the trustworthiness of peers. For example, a peer can develop a good reputation by being honest for numerous small transactions and then tries to make a profit by cheating for large transactions.
- Most systems do not provide incentives for a peer to rate others and suffer from insufficient feedback.
- Most systems cannot deal with strategic dynamic personality of peers. For example, malicious peers can build a reputation and then starts cheating or oscillating between building and milking the reputation.

In the following sections we present the design ideas of the trust framework, a coherent dynamic trust model, the strategies for developing system-level mechanisms to implement the model and a trust-based peer selection scheme, and the risk evaluation with the proposed approach, including how the above-mentioned problems can be avoided or reduced and how potential corrective and preventive methods can be used for recovery and survival.

2.2 The Trust Model

The main focus of this chapter is the design and development of a dynamic P2P trust model for quantifying and assessing the trustworthiness of peers in P2P e-commerce communities. A unique characteristic of our trust model is the identification of five important factors for evaluating the trustworthiness of a peer in an evolving P2P e-commerce community.
2.2.1 Trust Parameters

In PeerTrust, a peer's trustworthiness is defined by an evaluation of the peer it receives in providing service to other peers in the past. Such reputation reflects the degree of trust that other peers in the community have on the given peer based on their past experiences. We identify five important factors for such evaluation: (1) the feedback a peer obtains from other peers, (2) the feedback scope, such as the total number of transactions that a peer has with other peers, (3) the credibility factor for the feedback source, (4) the transaction context factor for discriminating mission-critical transactions from less or non-critical ones, and (5) the community context factor for addressing community-related characteristics and vulnerabilities. We now illustrate the importance of these parameters through a number of example scenarios.

Feedback in Terms of Amount of Satisfaction. Reputation-based systems rely on feedback to evaluate a peer. Feedback in terms of amount of satisfaction a peer receives during a transaction reflects how well this peer has fulfilled its part of the service agreement. Some existing reputation based systems use this factor alone and compute a peer $u$'s trust value by a summation of all the feedback $u$ receives through its transactions with other peers in the community. For example, buyers and sellers in eBay can rate each other after each transaction (+1, 0, -1) and the overall reputation is the sum of these ratings over the last 6 months. We can clearly see that these feedback-only metrics are flawed. A peer who has performed dozens of transactions and cheated 1 out of every 4 cases will have a steadily rising reputation in a given time duration whereas a peer who has only performed 10 transactions during the given time duration but has been completely honest will be treated as less reputable if the reputation measures of peers are computed by a simple sum of the feedback they receive. Dellarocas [26] concluded that binary reputation mechanisms will not function well and the resulting market outcome will be unfair if judgment is inferred from knowledge of the sum of positive and negative ratings alone.

Number of Transactions. As described above, a peer may increase its trust value by increasing its transaction volume to hide the fact that it frequently misbehaves at a certain rate when a simple summation of feedback is used to model the trustworthiness of peers. The
number of transactions is an important scope factor for comparing the feedback in terms of degree of satisfaction among different peers. An updated metric can be defined as the ratio of the total amount of satisfaction peer $u$ receives over the total number of transactions peer $u$ has, i.e., the average amount of satisfaction peer $u$ receives for each transaction. However, this is still not sufficient to measure a peer’s trustworthiness. When considering reputation information we often account for the source of information and context.

**Credibility of Feedback.** The feedback peer $u$ receives from another peer $v$ during a transaction is simply a statement from $v$ regarding how satisfied $v$ feels about the quality of the information or service provided by $u$. A peer may make false statements about another peer’s service due to jealousy or other types of malicious motives. Consequently a trustworthy peer may end up getting a large number of false statements and may be evaluated incorrectly because of them even though it provides satisfactory service in every transaction.

In PeerTrust we introduce the credibility of feedback as a basic trust building parameter, which is equally important as the number of transactions and the feedback. The feedback from those peers with higher credibility should be weighted more than those with lower credibility. We have developed two mechanisms for measuring the credibility of a peer in providing feedback. The concrete formulas will be discussed in Section 2.2.3.

**Transaction Context Factor.** Transaction context is another important factor when aggregating the feedback from each transaction as transactions may differ from one another. For example, if a community is business savvy, the size of a transaction is an important context that should be incorporated to weight the feedback for that transaction. It can act as a defense against some of the subtle malicious attacks, such as the example we mentioned earlier where a seller develops a good reputation by being honest for small transactions and tries to make a profit by being dishonest for large transactions. It can be seen as a simplified mechanism for more sophisticated risk management in E-Commerce [58]. In addition to using the value of the transaction, the functionality of the transactions is another important transaction context as one might trust another to supply books but not supply medical advice.
**Community Context Factor.** Community contexts can be used to address some of the community-specific issues and vulnerabilities. One example is to add a reward as a community context for peers who submit feedback. This may to some extent alleviate the feedback incentive problem. As another example, if a trust authority or pre-trusted peers (e.g., with digital certificate from the community) are available, then incorporating these community-specific context factors into the trust computation can make the trust metric more robust against certain manipulation of malicious peers.

### 2.2.2 General Trust Metric

We have discussed the importance of each of the five trust parameters. In this section we formalize these parameters, present a general trust metric that combines these parameters in a coherent scheme, and describe the formula we use to compute the values for each of the parameters given a peer and the community it belongs to.

Given a recent time window, let $I(u,v)$ denote the total number of transactions performed by peer $u$ with $v$, $I(u)$ denote the total number of transactions performed by peer $u$ with all other peers, $p(u,i)$ denote the other participating peer in peer $u$’s $i$th transaction, $S(u,i)$ denote the normalized amount of satisfaction peer $u$ receives from $p(u,i)$ in its $i$th transaction, $Cr(v)$ denote the credibility of the feedback submitted by $v$, $TF(u,i)$ denote the adaptive transaction context factor for peer $u$’s $i$th transaction, and $CF(u,t_k,t)$ denote the adaptive community context factor for peer $u$ during the period of $t_k$ and $t$. The trust value of peer $u$ during the period of time $t_k$ and $t$, denoted by $T(u,t_k,t)$, is defined in Equation 1.

$$T(u) = \alpha \sum_{i=1}^{I(u)} S(u,i) \cdot Cr(p(u,i)) \cdot TF(u,i) + \beta \cdot CF(u)$$

(1)

where

$\alpha$ and $\beta$ denote the normalized weight factors for the collective evaluation and the community context factor.

The metric consists of two parts. The first part is a weighted average of amount of satisfaction a peer receives for each transaction. The weight takes into account the credibility of feedback source to counter dishonest feedback, and transaction context to capture
the transaction-dependent characteristics. This history-based evaluation can be seen as a prediction for peer $u$'s likelihood of a successful transaction in the future. A confidence value can be computed and associated with the trust metric that may reflect the number of transactions, the standard deviation of the ratings depending on different communities. The second part of the metric adjusts the first part by an increase or decrease of the trust value based on community-specific characteristics and situations. The $\alpha$ and $\beta$ parameters can be used to assign different weights to the feedback-based evaluation and community context according to different situations. For instance, the $\alpha$ and $\beta$ values can be assigned properly so the trust value is set to be either the feedback-based evaluation when the peer has enough transactions and feedback or a default value otherwise.

Important to note is that this general trust metric may have different appearances depending on which of the parameters are turned on and how the parameters and weight factors are set. The design choices depend on characteristics of online communities. We argue that the first three parameters — the feedback, the number of transactions, and the credibility of feedback source are important basic trust parameters that should be considered in computation of a peer's trustworthiness in any P2P communities.

2.2.3 The Basic Metric

We first consider the basic form of the general metric as shown in Equation 2 by turning off the transaction context factor ($TF(u, i) = 1$) and the community context factor ($\alpha = 1$ and $\beta = 0$). It computes the trust value of a peer $u$ by a weighted average of the amount of satisfaction peer $u$ receives for each transaction,

$$T(u) = \sum_{i=1}^{I(u)} S(u, i) * Cr(p(u, i))$$ (2)

The feedback in terms of amount of satisfaction is collected by a feedback system. PeerTrust uses a transaction-based feedback system, where the feedback is bound to each transaction. The system solicits feedback after each transaction and the two participating peers give feedback about each other based on the transaction. Feedback systems differ with each other in their feedback format. They can use a positive format, a negative format, a
numeric rating or a mixed format. \( S(u, i) \) is a normalized amount of satisfaction between 0 and 1 that can be computed based on the feedback.

Both the feedback and the number of transactions are quantitative measures and can be collected automatically. Different from these two, the third parameter — credibility of feedback — is a qualitative measure and needs to be computed based on past behavior of peers who file feedback. Different approaches can be used to determine the credibility factor and compute the credible amount of satisfaction. One way is to solicit separate feedback for feedback themselves. This makes the problem of reputation-based trust management more complex. A simpler approach is to infer or compute the credibility value of a peer implicitly. We propose two such credibility measures in this chapter. The first one is to use a function of the trust value of a peer as its credibility factor so feedback from trustworthy peers are considered more credible and thus weighted more than those from untrustworthy peers. This solution is based on two assumptions. First, untrustworthy peers are more likely to submit false or misleading feedback in order to hide their own malicious behavior. Second, trustworthy peers are more likely to be honest on the feedback they provide. It is widely recognized that the first assumption is generally true but the second assumption may not be true at all time. For example, it is possible (though not common) that a peer may maintain a good reputation by performing high quality services but send malicious feedback to its competitors. In this extreme case, using a function of trust to approximate the credibility of feedback will generate errors. This is because the reputation-based trust in PeerTrust model is established in terms of the quality of service provided by peers, rather than the quality of the feedback filed by peers. We call the basic metric that uses the trust value of a peer recursively as its credibility measure PeerTrust TVM metric and it is defined in equation 3.

\[
T_{TVM}(u) = \sum_{i=1}^{I(u)} S(u, i) \times \frac{T(p(u, i))}{\sum_{i=1}^{I(u)} T(p(u, i))}
\]

(3)

The second credibility measure is for a peer \( w \) to use a personalized similarity measure to rate the credibility of another peer \( v \) through \( w \)'s personalized experience. Concretely, peer \( w \) will use a personalized similarity between itself and another peer \( v \) to weight the
feedback by \(v\) on any other peers. Let \(IS(v)\) denote the set of peers that have interacted with peer \(v\), the common set of peers that have interacted with both peer \(v\) and \(w\), denoted by \(IJS(v, w)\), is \(IS(v) \cap IS(w)\). To measure the feedback credibility of peer \(v\), peer \(w\) computes the feedback similarity between \(w\) and \(v\) over the common set \(IJS(v, w)\) of peers they have interacted in the past. If we model the feedback by \(v\) and the feedback by \(w\) over \(IJS(v, w)\) as two vectors, the credibility can be defined as the similarity between the two feedback vectors. Particularly, we use the root-mean-square or standard deviation (dissimilarity) of the two feedback vectors to compute the feedback similarity. This notion of local or personalized credibility measure provides great deal of flexibility and stronger predictive value as the feedback from similar raters are given more weight. It may also act as an effective defense against potential malicious collusions. Given the observation that peers in a collusive group give good ratings within the group and bad ratings outside the group, the feedback similarity between a peer \(v\) in the collusive group and a peer \(w\) outside the group will be low which will effectively filter out the dishonest feedback by peer \(v\) for peer \(w\). We call the basic metric that uses the personalized similarity as the credibility measure PeerTrust PSM metric and it is defined in equation 4.

\[
T_{PSM}(u, w) = \frac{\sum_{i=1}^{I(u)} S(u, i) \cdot Sim(p(u, i), w)}{\sum_{i=1}^{I(u)} Sim(p(u, i), w)}
\]

(4)

where

\[
Sim(v, w) = 1 - \sqrt{\frac{\sum_{x \in IJS(v, w)} (\frac{S(x, \hat{v})}{I(x, v)} - \frac{S(x, \hat{w})}{I(x, w)})^2}{|IJS(v, w)|}}
\]

(5)

Given that one of the design goals of the basic model is to emphasize on the roles of different trust parameters in computing trustworthiness of peers, in the rest of the chapter we will use the above two measures as examples and study their effectiveness, benefit and cost. We will study in Chapter 3 different similarity metrics as credibility measure in more depth.

2.2.4 Adapting the Trust Metric with Context Factors

We have discussed the motivations and scenarios for incorporating the adaptive context factors into our general trust metric. In this section we gave two examples of adapting the
metric using the transaction and community context factor respectively.

**Incorporating Transaction Contexts by Transaction Context Factor.** Various transaction contexts, such as the size, category, or time stamp of the transaction, can be incorporated in the metric so that the feedback for larger, more important, and more recent transactions can be assigned more weight than those for other transactions. For example, an adapted metric that incorporates the size of a transaction \( i \), denoted by \( D(u, i) \), is defined in Equation 6.

\[
T(u) = \sum_{i=1}^{I(u)} S(u, p(u, i)) \times Cr(p(u, i)) \times D(u, i)
\]  

(6)

**Providing Incentives to Rate by Community Context Factor.** Several remedies have been suggested to the incentive problem of reputation systems [73] such as market-based approaches and policy-based approach in which users will not receive rating information without paying or providing ratings. Implementing these approaches might stifle the growth of online communities and fledgling electronic markets. In PeerTrust, the incentive problem of reputation systems can be addressed by building incentives or rewards into the metric through community context factor for peers who provide feedback to others. For example, an adapted metric is defined in Equation 7 with a reward as a function of a ratio of total number of feedback peer \( u \) give others, denoted as \( F(u) \), over the total number of transactions peer \( u \) has during the recent time window. The weight factors can be tuned to control the amount of reputation that can be gained by rating others.

\[
T(u) = \alpha \times \sum_{i=1}^{I(u)} S(u, i) \times Cr(p(u, i)) + \beta \times \frac{F(u)}{I(u)}
\]  

(7)

### 2.3 Implementation Strategies

Although the trust model for a P2P community is independent of its implementation, the effectiveness of supporting trust in the community depends not only on the factors and metric for computing trust values, but also on the implementation and usage of the trust model in a P2P system. Typical issues in implementing a P2P trust model such as PeerTrust in a decentralized P2P network include decentralized and secure trust data management, i.e.
how to efficiently and securely store and look up trust data that are needed to compute the trust value of a peer, trust metric computation execution, and trust-based peer selection scheme. This section discusses the architecture, algorithm, and design considerations in implementing PeerTrust model in a decentralized P2P system.

2.3.1 Managing Trust Data: System Architecture

![System Architecture](image1)

![Data Location](image2)

**Figure 1: PeerTrust System Architecture**

Figure 1(a) gives a sketch of the system architecture of the basic trust framework. There is no central database. Trust data that are needed to compute the trust measure for peers are stored across the network in a distributed manner. The callout shows that each peer has a trust manager that is responsible for feedback submission and trust evaluation, a small database that stores a portion of the global trust data, and a data locator for placement and location of trust data over the network.

The trust manager performs two main functions. First, it submits feedback to the network through the data locator, which will route the data to appropriate peers for storage. Second, it is responsible for evaluating the trustworthiness of a particular peer. This task is performed in two steps. It first collects trust data about the target peer from the network through the data locator and then computes the trust value. We can see that the trust evaluation is executed in a dynamic and decentralized fashion at each peer. Instead of having a central server that computes each peer’s trust value, a peer obtains another peer’s
trust data from the rest of the peers and computes the trust value of this peer on the fly. This allows peers to get an up-to-date evaluation of the peer by other peers.

Different applications may use different data placement schemes, which determine how and where the data can be inserted, updated, and accessed. A number of P2P file sharing systems have emerged and each has its own data location scheme. Examples include Gnutella which use broadcast-based schemes and do not guarantee reliable content location, and CAN [71], Chord [83], Pastry [75], as well as P-Grid [2], which use a distributed hash table (DHT) to deterministically map keys into points in a logical coordinate space and guarantee a definite answer to a query in a bounded number of network hops, typically in the order of logN. Depending on the choice of a data location scheme, the implementation of the trust model may be somewhat different. Different schemes may also affect the overhead of the trust data management but should not affect the effectiveness of the trust metric. In the first prototype of PeerTrust, we use P-Grid primarily because we obtained the P-Grid source code. The trust data about a peer u, i.e. feedback u receives for each transaction are stored at designated peers that are located by hashing a unique ID of peer u to a data key. Each piece of feedback includes the following information: ID of peer u as the data key, timestamp or counter of the transaction, feedback for that transaction, ID of the peer who provides feedback, and other applicable transaction contexts. Each peer is responsible for multiple keys and maintains a routing table for other keys. When a peer receives a search or update request with a data key that it is not responsible for, it forwards the request according to its routing table. So the storage cost at each peer is proportional to the degree of replication and the amount of history to store. Figure 1(b) shows a simple example of a PeerTrust network of 6 peers constructed using P-Grid.

For such a data location scheme, there is a trust issue associated with it, namely, peers may misbehave by providing false data or random data when responding to a search request. Either majority voting or encryption can be used to address this issue. The data locator can be configured to have multiple replicas responsible for the same key. When a peer u is searching for trust data about another peer v, it finds all the replicas responsible for the key and combines the data using a majority voting scheme. An alternative is to incorporate
encryption techniques to enhance the security of the data location scheme, so that peers cannot tamper with the data on the routing path (see Section 2.3.5 for more detail).

2.3.2 Trust Computation

The trust evaluation component is responsible for computing the trust measure based on the reputation data that are collected about a peer. We propose two strategies for implementing each of the two basic trust metrics – PeerTrust TVM (Equation 3) and PeerTrust PSM (Equation 4). One is called dynamic trust computation (DTC), which uses fresh trust data collected at runtime to compute the trust value. The other is called approximate trust computation (ATC), which uses cache to speed up the trust computation process. We refer to the dynamic and approximate implementations of PeerTrust TVM as TVM/DTC and TVM/ATC and the two implementations for PeerTrust PSM as PSM/DTC and PSM/ATC respectively. We explain below how each of them is implemented.

**Dynamic Computation of PeerTrust TVM – TVM/DTC.** Recall PeerTrust TVM metric defined in Equation 3, it is clear that the metric defines a recursive function that uses the trust value of a peer as its feedback credibility measure. Peer $w$ needs to recursively compute other peers’ trust values as the credibility factor in order to compute peer $u$’s trust value. We implement the dynamic computation with iterative style. Given a column vector of trust values for $N$ peers, $N$ being the size of the community, the algorithm can simply start with a default trust value vector. As peer $w$ obtains feedback for each peer in the recent time window, denoted as $win$, it repeatedly computes the trust vector until it converges. When this is done, all trust values of the peers in the network will be available. A sketch of the algorithm is given in Algorithm 1.

**Approximate Computation of PeerTrust TVM – TVM/ATC.** It is obvious that dynamic computation is expensive as a peer needs to retrieve the trust data of all peers in the network even when it is only interested in evaluating the trustworthiness of a particular peer or a small subset of peers. Approximate computation provides a more cost-effective algorithm by using a trust cache at each peer. Each peer maintains a trust cache, denoted as $Cache_T$, which keeps the most recent trust values of other peers it has interacted with
Algorithm 1 ComputeTrust\_TVM/DTC(u)

Input: \( u \), Output: \( T(u) \)
for \( v = 1 \) to \( N \) do
  RetrieveFeedback\( (v, win) \)
  \( T^0(v) \leftarrow T_{\text{default}} \)
end for
repeat
  for \( v = 1 \) to \( N \) do
    \( T^{i+1}(v) \leftarrow \text{Compute equation 3 using } T^i \)
  end for
  \( \delta \leftarrow ||T^{i+1} - T^i|| \)
until \( \delta < \epsilon \)

in the past, Peer \( w \) only needs to compute the trust value of peer \( u \) when it cannot find \( u \)’s value in the cache.

When computing the trust value of peer \( u \), instead of dynamically computing the trust value of the peers who have filed feedback about \( u \) as the credibility factor, peer \( w \) looks for their trust values in the cache. It then uses the cache value in the case of a cache hit and simply uses a default value in a cache miss. It thus eliminates the recursive or iterative computation. Once peer \( w \) computes the trust value for \( u \), it adds the value to the cache.

A sketch of the TVM/ACT algorithm is provided in Algorithm 2.

Algorithm 2 ComputeTrust\_TVM/ATC(u)

Input: \( u \), Output: \( T(u) \)

\( Feedback \leftarrow \text{RetrieveFeedback}(u, \text{win}) \)
for \( i = 1 \) to Length\( (Feedback) \) do
  \( p(u, i) \leftarrow \text{feedback source of } Feedback(i) \)
  if \( Cache_T(p(u, i)) \neq \text{Null} \) then
    \( Cr(p(u, i)) \leftarrow Cache_T(p(u, i)) \)
  else
    \( Cr(p(u, i)) \leftarrow T_{\text{default}} \)
  end if
end for
\( T(u) \leftarrow \text{Compute equation 3} \)
\( Cache_T(u) \leftarrow T(u) \)

Dynamic Computation of PeerTrust PSM – PSM/DTC. Now we consider PeerTrust PSM metric defined in Equation 4 that uses a personalized similarity measure as the credibility to weight feedback from peers. When a peer \( w \) needs to evaluate the trustworthiness of another peer \( u \), peer \( w \) needs to compute the personalized feedback similarity between \( w \) and every other peer. Thus, it needs to retrieve not only the feedback about peer \( u \), but also all the feedback that are given by the peers who have had transactions with peer \( u \).
In algorithms that implement PeerTrust PSM metric, we have each peer also keep a local copy of the latest feedback given by itself in addition to the feedback about other peers it is responsible for so the storage cost is slightly more expensive than PeerTrust TVM implementations. The PSM computation is straightforward once the data are collected on demand. The algorithm proceeds as in Algorithm 3.

**Algorithm 3** ComputeTrust$_{PSM/ATC}$ executed at peer $w$

Input: $u$, Output: $T(u)$

1. $Feedback \leftarrow$ RetrieveFeedback($u$, $win$)
2. for $i = 1$ to $\text{Length}(Feedback)$ do
   1. $p(u, i) \leftarrow$ feedback source of $Feedback(i)$
   2. RetrieveFeedbackBy($p(u, i)$, $win$)
   3. $Cr(p(u, i)) \leftarrow$ ComputeSimilarity($w$, $p(u, i)$)
3. end for
4. $T(u) \leftarrow$ Compute equation 4

**Approximate Computation of PeerTrust PSM – PSM/ATC.** Similar to TVM/ATC, PeerTrust PSM/ATC provides a more cost-effective implementation of PSM metric. The main difference between PSM/DTC and PSM/ATC is how the credibility value is collected, dynamically on the fly or from the credibility cache. In PSM/ATC each peer maintains a trust value cache ($Cache_T$) and a credibility cache ($Cache_{Cr}$) to keep the trust values and credibility values the peer has computed in the past respectively. In the case of a miss in its trust cache, peer $w$ retrieves the feedback about peer $u$, looks up the credibility value of the peers who have provided feedback about peer $u$ in its credibility cache, computes the credibility value in case of a miss, adds the credibility value in the credibility cache, and finally computes the trust value and adds the trust value in its trust cache. Algorithm 4 gives a sketch of the PSM/ATC implementation.

Note that the extra storage cost that is needed in both ATC implementations for caching trust values and credibility values should be negligible as it only caches a single value for each peer. Assuming caching a trust value for one peer takes 1 unit of storage, say 1 byte, the extra caching cost in order to cache the trust values for all other peers in the network is $N - 1$ bytes. So the peer should be able to have a cache that can hold the trust and credibility values for all other peers in the network in most cases. Otherwise, it can use an LRU-like cache replacement policy to evict the least recently used data items from the
Algorithm 4 ComputeTrust$_{PSM/ATC}(u)$ executed at peer $w$

```
Input: u, Output: T(u)
Feedback ⇐ RetrieveFeedback(u, win)
for i = 1 to Length(Feedback) do
    p(u, i) ⇐ feedback source of Feedback(i)
    if Cache$_{Cr}$,(p(u, i)) ≠ Null then
        Cr(p(u, i)) ⇐ Cache$_{Cr}$,(p(u, i))
    else
        RetrieveFeedbackBy(p(u, i), win)
        Cr(p(u, i)) ⇐ ComputeSimilarity(w.p(u,i))
        Cache$_{Cr}$,(p(u, i)) ⇐ Cr(p(u, i))
    end if
end for
T(u) ⇐ Compute equation 4
Cache$_T$(u) ⇐ T(u)
```

cache when the cache is full. The cache should be refreshed periodically so the values reflect
the latest behavior of other peers. A straightforward scheme could be to refresh the value
after certain number of transactions and more sophisticated scheme can also be used.

2.3.3 Dealing with Dynamic Personality of Peers

The trust model we have discussed so far uses recent transactions and feedback to compute
the trust value of a peer. This is based on the following justification: when a peer’s reputation
is based on a cumulative average of his lifetime ratings, once that peer has established
a solid reputation, incremental ratings play little role in changing that reputation and thus
the peer has diminishing incentives to behave honestly. If, however, older ratings can be
discounted then a peer’s recent behavior always matters and the peer has continuing in-
centives to behave honestly [42]. However, more realistic malicious peers may adaptively
decide on a strategy in order to maximize their expected payoff given the rules of the game.

There are a number of ways in which such peers can attempt to fool the system and obtain
higher payoffs. For example, they can build a good reputation and then start cheating
occasionally at a rate that gives them a higher payoff but still allows them to maintain an
acceptable reputation. Or they could oscillate between periods of building and then milking
their reputation.

To address such potential dynamic behaviors of peers, we propose an adaptive time
window based algorithm in this chapter as a simple approach to better react to the above
behaviors. We will study the problem more formally in Chapter 4 and propose a formal
model to handle the oscillating behaviors in a more systematic way.

The basic idea of the time window based algorithm is to adaptively use a smaller time window to reflect the peer's most recent behavior when the peer is dropping its performance over a threshold. Concretely, when peer \( w \) is computing the trust value for peer \( u \), it first collects all feedback about \( u \) in the recent time window \( \text{win} \), and computes a trust value \( T_l \) using one of the four basic trust computation algorithms (TVM/DTC, TVM/ATC, PSM/DTC, and PSM/ATC). In addition, it computes another trust value \( T_s \) using a recent subset of the feedback taken by a time window \( \text{win}_s \) smaller than \( \text{win} \). The second value \( T_s \) will be returned as the final trust value of \( u \) if it is smaller than the first value by a certain threshold, which likely indicates the peer is dropping its performance recently. Otherwise, the first value \( T_l \) will be returned. A sketch of the adaptive time-window based computation method is given in Algorithm 5. By choosing a proper regular time window and adaptive window, this method makes the reputation of a peer hard to build, easy to drop, namely, the reputation cannot be quickly increased by a small number of good transactions but it will quickly drop if the peer starts cheating. Note that the adaptive algorithms are built on top of the four basic algorithms so dishonest feedback is handled by the respective basic algorithms. Regarding the computation cost, the only additional cost is to compute the second trust value \( T_s(u) \) and to run a condition test to determine if the adaptive trust value should be applied, which is minimal because it is using a subset of the trust data and credibility values that are already retrieved and computed in the computation for the first value \( T_l(u) \). In the rest of the chapter, we refer to the four basic algorithms as PeerTrust basic and the adaptive time window based implementation of the four basic algorithms as PeerTrust adaptive.

**Algorithm 5** ComputeTrustAdaptive\((u)\)

<table>
<thead>
<tr>
<th>Input: ( u ), Output: ( T(u) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feedback ( \Leftarrow ) RetrieveFeedback((u, \text{win}))</td>
</tr>
<tr>
<td>( T(u) ) ( \Leftarrow ) Compute trust value using Feedback</td>
</tr>
<tr>
<td>( T_s(u) ) ( \Leftarrow ) Compute trust value using a subset of Feedback taken by ( \text{win}_s ),</td>
</tr>
<tr>
<td>if ( T_l(u) - T_s(u) &gt; \epsilon ) then</td>
</tr>
<tr>
<td>( T(u) ) ( \Leftarrow ) ( T_s(u) )</td>
</tr>
<tr>
<td>end if</td>
</tr>
</tbody>
</table>

28
2.3.4 Secure Processing and Transmission of Trust Data

There are a number of known security threats due to P2P communication. We discuss in this section how to guarantee secrecy and integrity of the trust data obtained from other peers and the accountability of the peers providing such trust data.

The unauthorized manipulation of data can happen either in storage or during transmission. We use two layers of techniques, namely, PKI based scheme and data replication, to increase the security and reliability of the trust data management. The first layer is PKI based scheme. We require each peer to have a public and private key pair. Therefore a peer ID will be either a digest of its public key, obtained using a secure hash function, or the public key itself. For feedback submission, a peer \( v \) submits the feedback about peer \( u \), signed with its private key \( SK(v) \), along with its public key \( PK(v) \). The fact that each piece of feedback is signed with the feedback source’s private key guarantees the integrity of the feedback and the authenticity of the feedback origin. Even though peers may tamper with the data that are stored in its local database and provide false or random data when processing a search request later, other peers are able to detect whether the data is corrupted by verifying the signature and discard the data if it is corrupted. When a peer \( w \) wishes to evaluate the trustworthiness of peer \( u \), it issues a search request for peer \( u \)'s trust data, including in the request its public key \( PK(u) \). The peer responsible for the data, encrypt its response with \( w \)'s public key \( PK(w) \), sign it with its own private key, and send the signed encrypted response, together with its public key, to the polling peer. Upon receiving the response, peer \( w \) verifies the signature using the received public key, decrypts the message using its own private key \( SK(w) \). It then verifies the signature of each piece of feedback by the public key of the feedback source. The fact that the data are encrypted with the public key of the polling peer \( w \) protects their confidentiality. The fact that data are signed with the responding peer’s private key allows the detection of integrity violations of the data and the authenticity of their origin. Note that peers will not be able to selectively discard data during routing, as their content, being encrypted with the polling peer’s public key, is not visible to them.
With the above scheme, peers can still cause data loss by corrupting the data or selectively discard data that are stored in its local database. To combat this and data loss caused by other issues such as routing anomaly, data replication can be used at the second layer to improve the data availability and reliability. A secure trust computation algorithm proceeds as in Algorithm 6.

**Algorithm 6 ComputeTrustSecure(u) executed at peer w**

<table>
<thead>
<tr>
<th>Input: u, Output: T(u)</th>
<th>for j = 1 to r do</th>
</tr>
</thead>
<tbody>
<tr>
<td>response (\Leftarrow) RetrieveFeedbacksecure(u, PK(w), win)</td>
<td>Verify the signature of response using the attached public key</td>
</tr>
<tr>
<td>Feedback (\Leftarrow) Decrypt the data with its private key SK(w)</td>
<td>for i = 1 to Length(Feedback) do</td>
</tr>
<tr>
<td></td>
<td>Verify the signature of Feedback(i)</td>
</tr>
<tr>
<td></td>
<td>end for</td>
</tr>
<tr>
<td></td>
<td>T_j(u) (\Leftarrow) ComputeTrust(u, metric, computation)</td>
</tr>
<tr>
<td></td>
<td>end for</td>
</tr>
<tr>
<td></td>
<td>T(u) (\Leftarrow) Median(T_j(u))</td>
</tr>
</tbody>
</table>

### 2.3.5 Trust-based Peer Selection Scheme

A key objective of the trust-based peer selection scheme is to select one peer or a subset of peers that are most qualified to perform a job in terms of their reputation-based trustworthiness. The trust value produced by the trust metric gives a reputation-based trust measure. It can help peers to form a trust belief or action on other peers and to compare the trustworthiness of a set of peers. A higher value of \(T(u, t_k, t)\) indicates that peer \(u\) is more trustworthy in terms of the collective evaluation of \(u\) by the peers who have had transactions with \(u\) and other community context factors.

There are several usages of the trust value in P2P communities. First, a peer \(w\) can derive trust relationship with another peer \(u\) to determine whether to perform the next transaction or determining its pricing or negotiation strategies with peer \(u\). A decision rule is needed to derive a trust relationship based on the trust value and the situation. Each peer must consider to which degree the value of \(T(u, t_k, t)\) with the associated confidence value will make it trust \(u\) given a specific situation. A simple rule for peer \(w\) to form a trust action on peer \(u\) can be \(T(u) > T_{threshold}(w)\), where \(T_{threshold}(w)\) is the threshold trust value for peer \(w\) to trust another peer. The factors that determine the threshold \(T_{threshold}(w)\)
include how much peer \( w \) is willing to trust others, a manifest of dispositional trust [64], the extent to which an entity has a consistent tendency to trust across a broad spectrum of situations and entities. Other factors include the context of the potential transaction. For example, a more expensive transaction may require a higher threshold. More complex decision rules can be applied and are not our focus in this dissertation. Interested readers may refer to [58] for a number of models that derive a trust relationship from different parameters in an eCommerce environment.

A second usage is to compare the trustworthiness of a list of peers. For example, in a file sharing community like Gnutella, a peer who issues a file download request can first choose a set of potential peers from those respond to its request based on their connection speed, etc. Then it can compare the trustworthiness of the potential peers based on their trust value and choose the peer with the highest trust value to download the file. By doing this, it reduces the risk of downloading inauthentic or corrupted files from untrustworthy peers.

### 2.4 Experimental Evaluation

We performed initial experiments to evaluate the basic trust framework and show its feasibility, effectiveness, and benefits. The first one evaluates effectiveness of the framework in terms of its computation error against malicious behaviors of peers in two settings. The second one demonstrates the benefit of trust based peer selection scheme. The third one evaluates the effectiveness of the adaptive algorithm against dynamic personality of peers. Lastly, we evaluate the runtime overhead of different implementation strategies.

#### 2.4.1 Experiment Setup

We implemented a simulator in Mathematica 4.0 and this subsection describes the general simulation setup, including the community setting, peer behavior pattern, and trust computation.

**Community Setting and Behavior Patterns.** Our initial simulated community consists of \( N \) peers. We have one experiment with varying \( N \) to show the scalability of the trust framework and otherwise \( N \) is set to be 128. We also ran experiments with different number
of peers and it did not show an effect on the effectiveness of trust computation. The game
theory research on reputation introduced two types of players [27]. One is commitment
type or a long-run player who would always cooperate because cooperation is the action
that maximizes the player’s lifetime payoffs if the player could credibly commit to an action
for the entire duration. In contrast, a strategic type corresponds to an opportunistic player
who cheats whenever it is advantageous for him to do. We split peers into these two types
in our simulation, namely, good peers and strategic or malicious peers. The percentage of
malicious peers is denoted by $k$. We have one experiment with varying $k$ to show its effect
and otherwise $k$ is set to be 25%.

**Table 1: Basic Trust Framework: Experiment Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community Setting</td>
<td>$N$ # of peers in the community</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$k$ % of malicious peers in the community</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td>$mrate$ % of transactions a malicious peer acts malicious</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>$res$ % of peers who respond to a transaction request</td>
<td>5%</td>
</tr>
<tr>
<td>Trust Computation</td>
<td>$I_1$ # of transactions of peer $u$ in recent time window</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>$I_s$ # of transactions of peer $u$ in adaptive window</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>$n_{cache}$ # of cache units in ATC implementations</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>$r$ # of replicas of underlying DHT structure</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$nExp$ # of experiments over which results are averaged</td>
<td>5</td>
</tr>
</tbody>
</table>

The behavior pattern for good peers is to always cooperate in transactions and provide
honest feedback afterwards. While it is a challenging task to model peers’ malicious be-
behavior realistically, we start with two malicious behavior patterns to study the robustness
of PeerTrust approach, namely non-collusive setting and collusive setting. In non-collusive
setting, malicious peers cheat during transactions and give dishonest ratings to other peers,
i.e. give bad rating to a peer who cooperates and give good rating to a peer who cheats.
A malicious peer may choose to occasionally cooperate in order to confuse other peers and
fool the system. We use $mrate$ to model the rate that a malicious peer acts malicious. We
have one experiment varying $mrate$ to show its effect on trust computation effectiveness,
and otherwise $mrate$ is set to 100%. In collusive setting, malicious peers act similarly to the

32
non-collusive one, and in addition, they form a collusive group and deterministically help each other by performing numerous fake transactions and give good ratings to each other.

Two transaction settings are simulated, namely random setting and trusted setting. In random setting, peers randomly perform transactions with each other. In trusted setting, peers initiate transactions by issuing a transaction request. For each request, a certain percentage of peers respond. The response percentage is denoted by res and is set to 5% in the experiments. The initiating peer then uses the trust based peer selection algorithm to select the peer with highest trust value to perform the transaction.

Trust Computation. We use a binary feedback system where a peer rates the other peer either 0 or 1 according to whether the transaction is satisfactory. The number of transactions each peer has during the latest time window \( w_{in} \), denoted by \( I \), is set to be 100 for all peers. We evaluate the four algorithms – PeerTrust TVM/DTC, TVM/ATC, PSM/DTC, and PSM/ATC, as described in Algorithm 1, 2, 3, and 4 respectively. In the two ATC algorithms, the number of cache units (assuming storing a trust value for one peer takes one unit), denoted by \( n_{cache} \), is set to be \( N - 1 \). We also evaluate PeerTrust adaptive algorithm as described in Algorithm 5. The number of transactions each peer has during the smaller time window \( w_{in_s} \), denoted by \( I_s \), is set to be 20. We use P-Grid as the data location scheme to store and retrieve feedback data about peers. The degree of replication, denoted by \( r \), is set to be 4 in the experiments. For comparison purpose, we compare PeerTrust approaches to the conventional approach, referred to as Conventional, in which an average of the ratings is used to measure the trustworthiness of a peer without taking into account the credibility factor. All experiment results are averaged over 5 runs of the experiments. Table 1 summarizes the main parameters related to the community setting and trust computation. The default values for most experiments are listed.

2.4.2 Effectiveness against Malicious Behaviors of Peers

The objective of this set of experiments is to evaluate the effectiveness and robustness of the trust model against different malicious behaviors of peers. The experiments start as peers perform random transactions with each other. After 6400 transactions in the community,
i.e. an average of 100 transactions for each peer, a good peer is selected to evaluate the trustworthiness of all other peers. Each experiment is performed under both non-collusive and collusive settings described earlier. We compute the trust computation error as the root-mean-square (RMS) of the computed trust value of all peers and the actual likelihood of peers performing a satisfactory transaction, which is 1 for good peers and \(1 - mrate\) for malicious peers. A lower RMS indicates better performance.

![Graph](image1.png)

(a) Non-Collusive Setting  
(b) Collusive Setting

**Figure 2:** Trust Computation Error with Varying Percentage of Malicious Peers

![Graph](image2.png)

(a) Non-Collusive Setting  
(b) Collusive Setting

**Figure 3:** Trust Computation Error with Varying Malicious Rate of Malicious Peers

For the first experiment, we vary the percentage of malicious peers \(k\) and set the malicious rate to 1 \(mrate = 1\). Figure 2 represents the trust computation error of different PeerTrust algorithms and the conventional approach with respect to \(k\) in the two settings. Consider the non-collusive setting in Figure 2(a) first. We can make a number of interesting observations. First, we can see that the performance of the conventional approach drops almost linearly when \(k\) increases. Without taking into account the credibility of feedback
source, it is very sensitive to malicious peers who provide dishonest feedback. Second, both PeerTrust TVM/DTC and TVM/ATC stay effective when $k$ is less than 50%. Using trust values of peers recursively as the weight for their feedback, they are able to filter out dishonest feedback and make correct trust computations. However, the error becomes 100% when $k$ is greater than 50%, which indicates they completely make wrong evaluations by mistaking good peers as untrustworthy and malicious peers as trustworthy. This is particularly interesting because it shows that malicious peers are able to fool the system by overriding the honest feedback provided by good peers when they are the majority. We can also see that the performance of TVM/ATC and TVM/DTC are reasonably close. Last, both PeerTrust PSM/DTC and PSM/ATC stay effective even with a large percentage of malicious peers. This confirms that the personalized similarity based credibility acts as a very effective measure to filter out dishonest feedback. Also note that PSM/DTC and PSM/ATC give the same result when the system is stable. In the collusive setting in Figure 2(b), the interesting observations are both conventional approach and PeerTrust TVM approach are extremely sensitive to collusive attempts that dishonestly provide feedback even when the number of malicious peers is very small. On the other hand, PeerTrust PSM approach, as we have expected, acts as a very effective defense against collusion by filtering out dishonest feedback from the collusive group.

For the second experiment, we vary the malicious rate ($mrate$) and set the percentage of malicious peers to 25% ($k = 25\%$). Figure 3 represents the trust computation error of different PeerTrust algorithms and the conventional approach with respect to $mrate$ in the two settings. Again we can make a number of interesting observations in both settings. First, in the non-collusive setting (Figure 3(a)), the performance of the conventional approach drops when $mrate$ increases. Second, both PeerTrust TVM and PSM approaches have a slightly dropped performance when the malicious rate is less than 100%. This indicates that peers are able to confuse the system a little when they occasionally cooperates and gives honest feedback. The collusive setting (Figure 3(b)) shows similar results but to a larger extent.
2.4.3 Benefit of the Trust Based Peer Selection

This set of experiments demonstrates the benefit of the trust based peer selection scheme in which peers compare the trustworthiness of peers and choose the peer with the highest trust value to interact with. A transaction is considered successful if both of the participating peers cooperate. We define successful transaction rate as the ratio of the number of successful transactions over the total number of transactions in the community up to a certain time. A community with a higher transaction success rate has a higher productivity and a stronger level of security. The experiment proceeds by repeatedly having randomly selected good peers initiating transactions. In a community that has a trust mechanism, the source peer selects the peer with the highest trust value to perform the transaction. Otherwise it randomly selects a peer. The two peers then perform the transaction and the transaction succeeds only if the selected peer cooperates. The experiment is performed in both non-collusive setting and collusive setting. We show the benefit of PeerTrust approach and the conventional approach compared to a community without any trust scheme.

![Graph](image)

(a) Non-Collusive Setting  
(b) Collusive Setting

**Figure 4:** The Benefit of Trust Based Peer Selection Scheme

Figure 4 shows the transaction success rate with regard to the number of transactions in the community in the two settings. The graph presents a number of interesting observations. First, in the non-collusive setting (Figure 4(a)), we see an obvious gain of the transaction success rate in communities equipped with a trust mechanism. This confirms that supporting trust is an important feature in a P2P community as peers are able to avoid untrustworthy peers. Second, different trust mechanisms have different effectiveness. This
shows a similar comparison to the previous experiment. It is worth noting, however, that the conventional approach achieves a transaction success rate close to 100% even though its trust computation error is much higher than 0 shown in Figure 2(a). This is because even if the computed trust values do not reflect accurately the likelihood of the peers being cooperative but they do differentiate good peers from bad peers in most cases by the relative ranking. Last, in the collusive setting (Figure 4(b)), we can see that the transaction success rate is 0 for the conventional and PeerTrust TVM approach. This indicates that malicious peers are able to completely fool these trust schemes by collusion and render the system useless, even worse than the system without a trust scheme. However, the system still benefits from PeerTrust PSM approach significantly and shows robustness against the collusion.

2.4.4 Effectiveness against Dynamic Personality of Peers

So far we only considered peers with fixed personality in the above two settings. The goal of this experiment is to show how PeerTrust adaptive algorithm works against strategic dynamic personality of peers. Since we showed the effectiveness of PeerTrust basic algorithms against malicious behaviors of peers in providing dishonest feedback, we focus on the changing behaviors of peers without simulating dishonest feedback in this experiment. We simulated a community with all good peers but a malicious peer with dynamic personality. We simulated three changing patterns. First, the peer builds a reputation and then starts milking it. Second, the peer is trying to improve its reputation. Third, the peer oscillates between building and milking reputation. The experiment proceeds as peers randomly perform transactions with each other and a good peer is selected to compute the trust value of the malicious peer periodically. We compare PeerTrust adaptive algorithm in Algorithm 5 that uses an adaptive time window to PeerTrust basic approach that uses a fixed time window.

Figure 5 shows the computed trust value of peer \( u \) by PeerTrust adaptive metric and PeerTrust basic metric against different changing patterns. Figure 5(a) shows the computed trust value of the peer who is milking its reputation. We can see that, by using a time
Figure 5: Effectiveness against Dynamic Personality and Reputation Oscillation

window based metric that discounts the old feedback of peers, both PeerTrust basic and PeerTrust adaptive lead to a collapse of the reputation eventually. However, with PeerTrust basic, the peer is able to take advantage of the transition period by cheating while still maintaining a reasonable reputation before it completely collapses. On the other hand, PeerTrust adaptive is able to correct the trust value much sooner depending on how small the adaptive window is. Figure 5(b) shows the computed trust value of the peer who is building its reputation. The peer has to continuously perform good services for a period of time to achieve a good reputation from a bad one in both approaches. Figure 5(c) and 5(d) show the computed trust value of the peer who is oscillating between building and milking its reputation but with a different frequency. With PeerTrust basic, the peer gains by milking the reputation but it also has to pay the same amount of cost to build it back. In PeerTrust adaptive, a peer is quickly detected for its bad behavior but it cannot simply increase its trust value quickly by acting well for a short period so the cost of rebuilding reputation is actually higher than the gain of milking it.
2.4.5 Trust Evaluation Cost

The objective of this experiment is to understand the runtime overhead of different PeerTrust metrics and implementation strategies and how well it scales. Runtime overhead mainly comes from the cost of retrieving required information for each trust evaluation. It is proportional to two factors - the number of lookups and the cost of each lookup. The number of lookups varies with different trust metrics and implementation strategies. The cost of each lookup is determined by the underlying DHT scheme. As the lookup cost in DHT scheme is usually represented by the number of network hops or number of messages that are required in the routing process, we use the number of network hops for each trust computation as the metric for the runtime cost and compare the cost of different PeerTrust metrics and implementation strategies.

![Graph](image1.png)  
![Graph](image2.png)

(a) Trust Computation Scalability  
(b) Cost of Cache Bootstrapping

**Figure 6: Trust Computation Overhead**

We first compare the cost for the four PeerTrust algorithms with respect to the number of peers.

Figure 6(a) represents the cost with respect to the number of peers in the P2P community. Note that the graph uses a log-log scale. We analyze the result in detail. First, the two ATC approaches – PeerTrust TVM/ATC and PSM/ATC have the same cost and scales well. Consider peer \( w \) computing the trust value of peer \( u \). In these two algorithms, peer \( w \) only needs to retrieve the feedback about \( u \) and uses the cached values as the credibility value. As feedback about a peer is stored at multiple designated peers using the DHT structure, peer \( w \) issues multiple DHT lookups to get the data from all replicas and each
DHT lookup takes $O(lgN)$ network hops. So the cost is in the order of $lgN$ as well. Second, as expected, PeerTrust TVM/DTC requires $O(N)$ network hops for each computation does not scale well. Third, PeerTrust PSM/DTC requires higher lookup cost than PSM/ATC but it still scales well. In this algorithm, in addition to the feedback about peer $u$, peer $w$ also needs to retrieve the feedback that are given by the peers who have given feedback about peer $u$ in order to compute the similarity based credibility dynamically. As feedback by a peer is also stored locally by the peer itself, peer $w$ issues a direct lookup to every peer who have interacted with peer $u$. Specifically, since the number of transactions for each peer during the recent time window is set to be 100, so the difference of the cost between PSM/DTC and PSM/ATC, which is the distinctive number of peers who interacted with peer $u$ in recent 100 transaction, is close to 100 in most cases.

We also compared the lookup cost for TVM/ATC and PSM/ATC during the bootstrapping period when the trust cache and the credibility cache are being filled. Figure 6(b) represents the cost with respect to the number of trust computations performed. We can see that TVM/ATC does not have extra bootstrapping cost because it uses a default trust value as the credibility factor and the cache is filled as it computes trust values for other peers. On the contrary, PSM/ATC requires extra cost because it has to compute the credibility in the first few trust evaluations to fill the credibility cache. Once the credibility cache is filled, the cost becomes the same as TVM/ATC.

2.5 Summary

We have presented a basic reputation-based trust supporting framework as a mechanism for alleviating or resolving some of the security problems in P2P communities by choosing reputable peers and avoid untrustworthy peers. For example, the simplest version of a virus attack or a DOS attack in a file sharing system would be that an adversary responds to a file request with a fake file name and delivers a virus or bogus content to flood the disks. With a reputation based trust mechanism in place, the peer who receives the malicious content will be able to submit a negative feedback about the malicious peer and help other peers to avoid it in the future.
The framework includes a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers based on a transaction-based feedback system, and a decentralized implementation of such model over a structured P2P (P2P) overlay network. We reported initial simulation-based experiments, demonstrating the feasibility, effectiveness, and benefits of our approach.

Not surprisingly, a reputation-based trust mechanism also introduces vulnerabilities and problems by itself. Common attacks are known as shilling attacks where adversaries attack the system by submitting fake or misleading ratings to confuse the system. A shilling attack is often associated with a pseudo-spoofing attack, where one identity creates multiple pseudonyms to boost each other’s ratings, or collusion among peers, where a group of malicious peers collaborate to raise each other’s rating and to badmouth other peers. Further, peers can amount attacks on the trust management system by distributing tampered with trust information.

The basic trust framework we have described in this chapter tries to minimize such security weaknesses. For example, the use of the credibility factor of the feedback source can be seen as an effective step towards handling fake or misleading ratings in reputation-based feedback. The ability to incorporate various transaction and community contexts can also act against some of the subtle attacks. Furthermore, by combining the proposed trust metric and the secure trust data transmission built on top of mature public key cryptographic algorithms, it prevents distribution of tampered with trust information and man in the middle attack. An identity is established by public key that corresponds to a secret private key. Therefore, each identity cannot be spoofed without the knowledge of the corresponding private key. Any content properly signed will not have its integrity or origin compromised.

In the following chapters, we will focus on some of the vulnerabilities and identify threat models and develop countermeasures to make the framework more dependable to various malicious behaviors of peers.
CHAPTER III

COUNTERING FEEDBACK SPARSITY

We have presented the basic trust framework in the previous chapter as a mechanism for alleviating or resolving some of the security problems in P2P communities by choosing reputable peers and avoid untrustworthy peers. A number of reputation systems have also been proposed recently for P2P networks [3, 23, 49, 95, 60, 101]. The core algorithmic component of a typical reputation system is to aggregate community feedback about peers’ past behaviors into a reputation trust value as a measure of trustworthiness for each peer. However, a reputation-based trust mechanism also introduces vulnerabilities and problems by itself and few of the reputation management works so far have focused on the vulnerabilities of a reputation system itself.

In the rest of the dissertation, we refine the basic trust framework and propose a set of dependable trust evaluation mechanisms to counter various malicious behaviors of peers. We define threat models and develop defense mechanisms that enable the trust framework to be resilient to various vulnerabilities, including risks involved in the presence of sparse and dishonest feedbacks, strategic oscillating behaviors of malicious participants, and the loss of feedback privacy.

In this chapter, we explore the risks involved in the presence of sparse and dishonest feedbacks and propose a similarity based inference framework to tackle the feedback sparsity and integrity issues.

3.1 Introduction

As the task of a reputation system is to aggregate community feedback about peers’ past behaviors into a reputation trust value as a measure of trustworthiness for each peer, a major challenge of designing a reputation system is to cope with the sparsity of the feedback and potential manipulations of the feedback.
• **Feedback Sparsity.** An important problem with reputation systems that has caught less attention so far is the sparsity of the feedback data. Reputation systems rely on historical transaction and feedback data to derive a reputation trust score for entities. In large scale peer-to-peer networks, each peer may have interacted with only a small number (percentage) of peers and hence has few feedbacks about and from other peers. It is been shown that limited reputation sharing or no reputation sharing when personal experiences are available [60] helps minimize the adverse effect of feedback attacks by malicious peers. However, when the feedback data is sparse, it is often not possible for peers to have personal experiences with each other. As a result, the problem of feedback sparsity has a major negative impact in reputation systems. When the feedback matrix is sparse, the system cannot evaluate the trust value most peers or the output evaluations suffer from accuracy.

• **Feedback Manipulation.** A piece of feedback is simply a statement from a user about another user, typically, the service consumer about the service provider in a peer-to-peer network. There is no mechanism to guarantee that the statement is honest. Malicious users may manipulate the feedback data in a way in order to benefit themselves or damage the system. A main technique in the reputation systems proposed so far is to associate a credibility weight with each peer (as versus the service reliability of the peer). Theoretically, a peer who has consistently provided honest feedback will be associated with a higher credibility weight so the system can differentiate honest feedback from dishonest ones. We have shown in Chapter 2 that the user similarity based on previous feedbacks about common peers can be used as a personalized credibility measure in the trust metric and is very effective against certain attacks including collusions among a group of peers who provide good ratings within the group and bad ratings outside the group. Unfortunately, when the feedback is sparse, it is highly probable that two users have not interacted with any common set of peers and thus the feedback based similarity between two users cannot be computed.
Interestingly, the problem of feedback sparsity and manipulation are closely related. Feedback sparsity concerns with the quantity of feedback data. Feedback manipulation concerns with the quality of feedback data and in most cases may affect the quantity of good quality feedback data. The two problems also have a magnifying effect on each other. On one hand, when adversaries pollute the feedback data, the valid or usable feedback data becomes sparse, which makes sparsity problem worse. On the other hand, when feedback data is sparse, the feedback related attacks may be magnified and have a more detrimental effect.

We devote this chapter to study feedback sparsity problem with potential feedback manipulations and explore the potential solutions. Research in trust inference [96, 13, 84, 37] addresses trust propagation of initial trust relationships assuming certain transitivity of trust. They differ with reputation systems in that they assume the initial trust relationship is predefined among nodes and the proposed approach typically uses graph theoretic models or matrix operations to propagate the initial trust relationships. Some recent works adopted certain inference techniques in reputation systems. A recent notable work EigenTrust [49] derives initial trust based on personal feedback and perform a global trust propagation till it finds the Eigenvector of the initial trust matrix. However, it has been shown that the model is vulnerable to certain types of attacks where adversaries exploit the trust transitivity property by creating false trust links.

While these works shed lights on the potential adoption of inference techniques to address sparsity issues in reputation systems, a few research challenges remain. First, how do we derive the initial relationships among nodes and what information or relationships do we use to perform inference so that it is robust to potential user manipulation? Second, how do we design the inference model and select the right parameters such as depth of inference and inference functions to deal with sparse data while on the other hand to avoid unnecessary cost. Finally and importantly, how does the inference model cope with feedback related vulnerabilities in the reputation system?

Bearing these questions in mind, we extend the basic trust framework and propose a similarity inference based scheme and study experimentally how different inference parameters
and models perform in countering sparsity and vulnerabilities.

This chapter has a few unique contributions. First, we explore different attack strategies related to feedback vulnerabilities and also formally define the sparsity problem. Second, we propose a similarity inference based scheme with algorithmic details to counter the feedback sparsity and integrity issues. Finally, we study how variant algorithmic component of the framework perform in coping with feedback sparsity and vulnerabilities.

The rest of the chapter is organized as follows. Section 3.2 presents the sparsity problem and describes the threat model by exploring different attack strategies. Section 3.3 presents a similarity inference scheme with variant algorithmic techniques. Section 3.4 experimentally studies how the different algorithmic components help coping with the feedback sparsity and vulnerabilities. Finally, Section 3.5 summarizes the chapter.

3.2 Problem Statement

We first define or review certain terms that will facilitate our discussion and comparison of various reputation schemes. We then examine the problem space for feedback sparsity as well as feedback integrity by defining the threat model and attack strategies.

3.2.1 Terms and Definitions

Peer-to-peer community. The peer-to-peer community consists of \( N \) peers who perform transactions with each other. The community can be built on top of a client server network or a pure P2P network. It can be also generalized into an online community that consists of \( N \) users and \( M \) service providers. In P2P community, the number of users and service providers happen to be the same as each peer serve both as a client and a server.

Transaction. The community is defined by interactions or transactions among peers. These transactions may include downloading files, storing data, or monetary transactions. Each transaction has a client (service consumer) and a server (service provider) and each peer may serve as a client in one transaction and as a server in another.

Transaction feedback. A transaction feedback is a statement issued by the client about the quality of a service provided by the server in a single transaction. This can be collected explicitly or derived implicitly.
**Personal feedback.** A personal feedback or opinion is a user’s general impression about a service provider based on its personal experiences with the server. It can be derived from its feedback on all the transactions that are conducted with the server in the past, e.g. as the percentage of positive transactions or the average rating for each transaction. We can represent all the personal feedback in the network as a user-user opinion matrix, where each cell represents an opinion from a given user about a given server.

**Reputation Trust.** The goal of a reputation system is to compute a reputation trust for a given service provider from a given user’s perspective. We refer to the evaluating user as *source* and the user to be evaluated as *target*. A reputation trust matrix is a user-user matrix, where each cell represents a reputation trust score for a given service provider from a given user’s perspective. We call the reputation trust global if the reputation trust for each service provider is the same across all the users. Otherwise, it is personalized.

In some trust schemes, if a user *u* has direct interactions with a server *s*, then *u*’s rating of *s* is treated as *s*’s trustworthiness from the *u*’s point of view. In PeerTrust as well as some other schemes, *u* still needs to consider the other users’ ratings about *s* when evaluating *s*’ trustworthiness. A peer’s trustworthiness is the combination of the user’s personal opinion and the community feedback. Thus we differentiate the user rating from trustworthiness.

**Credibility.** Credibility of a user *u* indicates how credible *u* is in providing feedback or rating is in general. In contrast, trustworthiness indicates how reliable *u* is in providing service. Some works use trustworthiness as a general notion. In PeerTrust, we argue that credibility should be differentiated from trustworthiness (reliability). User *a* may trust user *b* for its ratings or recommendations but not necessarily its service. It is also referred to as recommendation trust in some literature [13] as versus service trust. We will use the term credibility for this purpose and reserve the term trust for the reputation based service trust (reliability) in this dissertation. Credibility of a peer can be collected or defined explicitly or derived computationally.
3.2.2 Threat Model

Reputation systems have to cope with potential manipulations from adversaries. A common goal for malicious users is to boost their own ratings or decrease the ratings of other peers by manipulating feedbacks so they can perform malicious services when other peers select to perform a transaction with them. Other different incentives include free riding. We do not consider this type of behaviors in this dissertation and interested readers may refer to [4, 70, 100].

A feedback manipulation attack corresponds to the addition of noise to the feedback data (training data). A common strategy for adversaries is to provide false ratings. We consider various dimensions of these types of attacks in this section.

Attack Goals. Based on different goals of the attacks, we consider the following two types of attacks.

- **Random attacks.** The goal of this type of attacks is to reduce the overall performance of a system as a whole. They are not directed at any particular users.

- **Target attacks.** These are attacks that try to force the ratings for a target user to a particular target value. For example, in a nuke attack, the goal is to force all predicted ratings of targeted users to the minimum rating. Similarly, in a push attack, the goal is to force the ratings to the maximum rating.

Attack Models. We also consider two different types of attacks models.

- **Non-Collusive.** In the non-collusive model, individual malicious users do not know each other and they each do something bad and hoping it will affect the system.

- **Collusive.** In the collusive model, multiple malicious peers may form a group and collude with each other in order to achieve their goal. The goal is typically targeted towards boosting the ratings of the whole or part of the group.

A related attack is for an individual user to create multiple fake profiles and act as a collusive group. We will treat this same as the collusive attack. We do assume
adversaries have to pay a cost to create a profile so it is not feasible to create infinite false profiles. The goal of designing a robust algorithm is to maximize the noise level it can tolerate so an adversary has to pay a high cost in order to achieve their malicious goal.

**Attack Strategies.** Some attacks may be specifically designed to exploit a particular weakness in a specific algorithm or class of algorithms. For example, for systems that does not differentiate trust (reliability) and credibility, one simple strategy for the malicious group is to have part of the peers act as front peers or moles [61, 31]. These peers always cooperate with other peers in order to increase their reputation and then provide misinformation to promote other malicious peers. We will discuss them in more detail when we review different reputation systems in Chapter 7.

On the other hand, differentiating trust (reliability) and credibility helps systems to avoid the front peer attacks and are more robust to dishonest feedbacks. In Chapter 2, we have studied a non-collusive model and a nave collusive model and have shown that using user similarity as a credibility weight when aggregating community feedback provides promising results in defending against dishonest feedback attacks. So in the rest of this chapter, we will focus on the similarity based PeerTrust models and explore more sophisticated attack strategies and the sparsity problem and develop enhanced framework to defend against them.

As malicious users do not collude with each other in non-collusive setting and the individual attack strategy is straightforward, we focus on the attack strategies in the collusive model. Assuming the goal for the group is to boost the ratings for the group. There are different strategies that the users may use in rating other peers that are outside their group. Similar to the shill attack designs from [54] in a collaborative filtering context, we study the following strategies,

- **Collusive.** A straightforward way is for each collusive user to rate each other in the group with the maximum rating and rate peers outside the group with a minimum rating. The hope is while they boost their own ratings, they also decrease other peers'
ratings or damage the performance of the reputation system. However, by doing this, they may end up with having very low similarity to the normal users and in turn their ratings will not be counted as much.

- **Collusive Camouflage.** A more sophisticated attack is for the malicious users to rate each other within the group with maximum rating but rate peers outside the group honestly so that they will be similar to more existing honest users, and thus, have a larger effect on boosting their own ratings. Note that this strategy is not considered in the basic PeerTrust framework. Hypothetically, this strategy will allow adversaries to amount a more detrimental attack and boost their ratings by camouflaging as honest users.

### 3.2.3 Sparsity Problem

Now we consider the sparsity problem for using such a similarity based trust scheme. We first consider the basic PeerTrust PSM metric 4. The metric computes the reputation trust of a user as a weighted average of previous ratings about the user. The credibility weight is a personalized similarity measure between the evaluating peer and the peer who provides ratings. Concretely, peer w will use a personalized similarity between itself and another peer v to weight the feedback by v on any other peers. We define the similarity by modeling the feedback by v and the feedback by w over a common set of peers for which both v and w have rated as two vectors and computing the similarity between the two feedback vectors. Particularly, we use the root-mean-square or standard deviation (dissimilarity) of the two feedback vectors to compute the feedback similarity.

It is important to note that the similarity is defined over the common set of peers that both peers have rated. When the input ratings matrix is sparse, two given peers may have a very limited number or zero number of co-rated peers and the similarity can not be derived. As a result, the reputation trust can not be computed.

Now we formally analyze what is the probability of an undefined prediction given a sparse feedback matrix. Given a source user a and a target service provider j, let \( U_j \) denote the set of users who have rated \( j \), let \( N_a \) denote the set of neighbors of \( a \). \( N_a \) is selected
by the neighborhood selection technique out of all the users who have co-rated items with a. The basic PSM metric computes the reputation trust by aggregating the feedback from those neighbors of a who have rated j computed as the intersection between $U_j$ and $N_a$.

When the input matrix is sparse, it has three effects. First, $a$ will have a limited number of neighbors, i.e. $N_a$ is small, as there is less chance that other users will have co-rated items with $a$. Second, $j$ will have a limited number of ratings, i.e. $U_j$ is small. As a result, the probability of the intersection of $N_a$ and $U_j$ being empty will be high which results in an undefined prediction.

Suppose each user rates $r$ users among the other $n-1$ users. The average number of ratings per user is also $r$. For simplicity, we assume all users who have co-rated items with $a$ are considered as a neighbors and no neighborhood selection is used. In this case, the probability of a user not belonging to $a$‘s neighborhood is the probability that it does not have co-rated item with $a$. The probability of an undefined prediction is the probability that all of the users who have rated item $j$ do not belong to $a$‘s neighborhood. So the final probability is given by Equation 8.

$$p(n, r) = \left( \frac{n-r}{n} \right)^r$$

In a large open community, most users will only interact with a small set of users and may provide ratings to an even smaller set of users among those. The general sparsity problem refers to such situation where while $n$, the number of users, could be of the order of thousand, $r$, the number of ratings per user, could be very small. The resulting probability of undefined prediction can be high. When a new user joins the community, it does not have many ratings about other users nor does it have many ratings from other users. Suppose a new user $a$ only has $r_a$ ratings ($r_a \ll r$). The probability of undefined prediction becomes $\left( \frac{n-r_a}{n} \right)^r$. Even though the input ratings matrix may not be sparse in general, the probability is still fairly high due to a small $r_a$. This is sometimes referred to as cold start problem.

### 3.3 A Similarity Inference Scheme

Bearing the above analysis in mind, we propose an inference scheme based on similarity weight to provide sparsity resilience and robustness.
3.3.1 Overview

We first model our problem in a graph theoretic way. Recall that in our personal rating matrix, each cell represents a personal rating from a node about another node. Suppose each peer is represented a node in the graph. If peer \( u \) has interacted with and has a personal rating about peer \( v \), it corresponds to a directed link from \( u \) to \( v \). Each peer then has a set of outgoing links representing the ratings it has given to other peers and a set of incoming links representing the ratings it has received from other peers. Figure 7 illustrate a partial personal rating matrix and a corresponding graph with solid links representing the personal rating links.

(a) Personal Rating Matrix       (b) Rating and Neighborhood Graph

![Diagram](image)

**Figure 7:** Similarity Inference Scheme: Illustration

When peer \( u \) is evaluating the trustworthiness of peer \( v \), it can use its personal rating if it is available (direct link from \( u \) to \( v \)) or the community reputation based on the community ratings about peer \( v \) (all incoming links to \( v \)). Without loss of generality, in our framework, the trust value is defined by a linear combination of the user’s personal rating and the community reputation. The user can tune the weight if it desires to give more weight to its personal experiences when it is available. In the rest of the chapter, we will focus on the computation of the community reputation.

The community reputation aggregates all the ratings about \( v \). As we have discussed earlier, a common strategy to combat the dishonest feedback is to use certain metric to weigh the ratings during aggregation. There have been a few classes of metrics that have
been used. One common way is to use the peer’s previously determined reputation score to weigh its feedback. However, as we have discussed earlier, a user may have personal experiences with another user and get opinions on the user for performing service but it does not indicate any opinions on the user for providing feedback. It has been shown these schemes are vulnerable to certain types of attacks where adversaries exploit the transitivity property of the trust links and create fake links.

We argue that we need to derive a separate credibility link between users so they can use them to weigh each other’s ratings. Intuitively, a user is more willing to trust another user’s ratings if they have given similar ratings to the same set of peers that you have done before. If we treat all the ratings each peer has given to others, we can derive certain relationship between users in terms of their feedback filing and use this as a credibility measure. As a result, in addition to the personal rating links, we add another type of links, credibility links, with each link representing the similarity between a pair of users. The credibility links are illustrated as dashed links in Figure 7.

For some pair of users, the similarity can be directly derived using certain similarity measure between the ratings they have given before over a common set of peers. And we will discuss in detail different similarity measures and study experimentally how they affect the performance. However, when the input data is sparse, it is not always possible to derive the direct similarity or the derived link may be unreliable because they are based on too little information.

This motivates us to explore the transitive associations of similarity. If a user $u$ is similar to $v$, $v$ is similar to $w$, $u$ should be somewhat similar to $w$. In other words, similarity may propagate through the network. The indirect similarity link is represented as double dashed link in Figure 7. This is particularly important when the input feedback matrix is sparse. We will present algorithms that help peers find indirect neighbors and compute indirect similarity weight for indirect neighbors.

In summary, the framework uses user feedback similarity to weigh their ratings and uses similarity inference to compute the weight for indirect neighbors. It has the following main steps.
• Compute initial trust based on personal experience. The result of this is the personal experience matrix.

• Compute direct similarity based on the ratings matrix.

• Find indirect neighbors and compute indirect credibility.

• Aggregate user feedback (possibly from a selected neighborhood of users) based on the credibility of the users.

There are a number of research questions we would like to study in the proposed scheme. First, what kind of similarity do we use? Second, what kind of inference model do we use and how do we determine the depth of the inference and the inference functions? Third, do we use all the user’s neighbors or use a subset of the neighbors and what criteria? Finally, how do all the algorithm variants handle the sparsity issue as well as the attacks?

In the rest of the section, we present the details of each component of the scheme and study these questions by examining various algorithms and techniques and their effects on countering sparsity and vulnerabilities.

3.3.2 Computing Credibility using Similarity

We first derive the weight of direct links between two users. Assuming the personal feedback $v_{a,i}$ is derived as an average of all the transaction feedback (satisfaction) from $a$ to $i$ in previous transactions. If we treat all the ratings each peer has provided in the past about other peers as a vector or variable, we can compute similarity between two users using different measures. The intuition is that a user will trust another user’s ratings if the latter has given similar ratings as the user does previously. Depending on different similarity measures, the direct similarity can be undefined. We adopt the following three measures in our framework, including the one we used in Chapter 2.

• Pearson correlation. Pearson correlation is widely used in collaborative filtering systems. It computes the degree of linear relationship between two variables. It ranges
from +1 to -1 where +1 means there is a perfect positive linear relationship. The Pearson correlation coefficient between user $a$ and $b$ is calculated as:

$$w_{a,b} = \frac{\sum_{i \in I_a \cap I_b} (v_{a,i} - \bar{v}_a)(v_{b,i} - \bar{v}_b)}{\sqrt{\sum_{i \in I_a \cap I_b} (v_{a,i} - \bar{v}_a)^2(v_{b,i} - \bar{v}_b)^2}}$$  \hspace{1cm} (9)

- Vector cosine similarity. The vector similarity measure is another widely used technique in which each user is represented by a vector of ratings in the $|I|$-dimensional space, where $I$ is the set of items. The similarity between two users $a$ and $b$ is calculated as the cosine of angle between the two corresponding vectors (normalized inner product):

$$w_{a,b} = \frac{\sum_{i \not\in I} v_{a,i}v_{b,i}}{\sqrt{\sum_{i \in I_a} v_{a,i}^2} \sqrt{\sum_{i \in I_b} v_{b,i}^2}}$$  \hspace{1cm} (10)

- Euclidean distance. In the previous chapter, we have also used the root-mean-square or standard deviation (dissimilarity) of the two feedback vectors to compute the similarity. It is calculated as:

$$w_{a,b} = 1 - \frac{\sqrt{\sum_{i \in I_a \cap I_b} (v_{a,i} - v_{b,i})^2}}{|I_a \cap I_b|}$$  \hspace{1cm} (11)

An inherent difference between the Euclidean distance and the other two is that it does not perform a normalization of the feedback. Intuitively, normalization of feedback will minimize the effect of user bias. For instance, one user may always give high ratings for all other users while another may always give low ratings. When the feedback is normalized, it is the relative ratings that matter and these two users will have a high similarity despite their bias. On the other hand, the Euclidean distance will treat these two users as dissimilar.

Other potential measures include Spearman correlation which takes the ranks of the two variables and computes a ranked version of Pearson correlation. Existing collaborating filtering literature [17, 41] has reported that Pearson correlation and Spearman correlation yield comparable and slightly better accuracy than vector cosine similarity in recommender systems typically evaluated by movie rating datasets. We will study experimentally how
these measures perform in our context with sparse feedback data and potential feedback manipulations in Section 3.4.

Significance Weighting. Similarity measures based on the co-rated users, such as Pearson correlation, give unreliable measures when the number of co-rated items is small. The more data points that we have to compare the opinions of two users, the more we can trust that the computed correlation is representative of the true correlation between the two users. Significance weighting is a technique that adds a correlation significance weighting factor to devalue similarity weights that were based on a small number of co-rated items. Equation 12 shows a modified similarity weight based on the number of common items. It has been shown that applying the significance weighting increased the accuracy of the prediction algorithm in collaborative filtering systems by a relatively large amount [41]. For example, a significance weighted similarity measure is shown below.

\[ w_{a,b} = w_{a,b} \times \max(1, \frac{n_{a,b}}{N}) \]  

(12)

where \( n_{a,b} \) is the number of co-rated items user \( a \) and \( b \) have and \( N \) is a threshold.

3.3.3 Similarity Inference

We refer to all the users who have a direct similarity weight defined with \( u \) as \( u \)'s direct similarity neighbors. Once the direct similarity neighborhood is formed, when \( u \) needs to evaluate another service providers trustworthiness, it asks its neighbors for their opinions about the target peer and aggregate their opinions. (The user can either use all its neighbors or select a subset of neighbors. We will discuss neighborhood selection techniques later in this section). However, when the input data is sparse, each peer may have a very limited number of direct neighbors and thus the chance increases that none of its direct neighbors has an opinion about a given target peer. In this case, it is natural to consider the neighbors' neighbors in order to get a larger neighborhood for future evaluations. In this section, we explore the transitive associations between similarity and present algorithms to find indirect neighbors of a given user and propagate the similarity weight between the user and indirect neighbors.

Discovering Indirect Neighbors

55
Figure 8: Discovering Indirect Neighbors: Illustration

We can model the problem as a transitive closure problem to find all nodes that are reachable by node $u$ by at least one path. (Note here that the path consists of only similarity links). Intuitively, a user may be only willing to take the opinions of other users that are within certain distance. An indirect neighbor that is too far away on a similarity path stops being useful even if each node is perfectly similar to their immediate neighbors on the path. So we bound the path by a maximum path length $L$ meaning we only consider the indirect neighbors within $L$ hops of the source user $u$. If a node is reachable by $u$ by at least $l$ hops, we call the node $u$’s $l$-hop neighbor. $u$’s 1-hop neighbors are also $u$’s immediate neighbors. Figure 8 shows the 1-hop (immediate an illustration of the similarity based neighbors.

Since we have a maximum path length bound, we do not need a full transitive closure algorithm. Instead, for each node, similar to the single source transitive closure problem, we can use a standard breath first graph search algorithm to find all the indirect neighbors that are within $L$ hops. Since the input data may be sparse and each user only has a limited number of immediate neighbors, we expect the algorithm to be efficient.

Computing Indirect Weights The similarity weight for $u$’s immediate neighbors are the direct link weight. For the neighbors greater than 1-hop, we need to compute an indirect weight for them. For a $l$-hop neighbor $v$, we consider all the $l$-length paths from $u$ to $v$ and compute an indirect weight based on these paths.

Formally, for each path, we infer the indirect weight for this path as a product of the direct weight on each link as in Equation 13. We can also add in a decaying factor at each
step such that the similarity decays at each step and essentially a path decays to 0 if it is too long. Since we have bounded the path length by a maximum length in finding neighbors, we set the decay factor as 1 in our implementations.

\[ p^{i}_{a,j} = w_{a,u_1} \ast w_{u_1,u_2} \ast \ldots \ast w_{u_{l-1},u_l} \ast v_{u_l,j} \]  

(13)

If there are multiple paths of length \( l \) from node \( u \) to node \( v \), we need a function to select or combine the inferred values from each path into a single value. We consider three such functions: maximum, minimum, and average.

- **Maximum Value.** Maximum function assumes an optimistic behavior of the user and selects maximum similarity value of the multiple paths. This is consistent with performing a generalized or operation over [0,1] valued beliefs in fuzzy logic. User \( u \) will believe another user \( j \) to an extent with which at least one of its closer neighbors trust \( j \).

\[ s_{a,j} = \min_{i} p^{i}_{a,j} \]  

(14)

- **Minimum Value.** Minimum function assumes a pessimistic behavior of the user and selects minimum similarity values of the multiple paths. This is consistent with performing a generalized and operation in fuzzy logic. User \( u \) will only believe another user \( j \) to an extent with which all its closer neighbors trust \( j \).

\[ s_{a,j} = \min_{i} p^{i}_{a,j} \]  

(15)

- **Average Value.** Average function computes an average of the similarity values of the multiple paths. Intuitively, this may give us the best of maximum and minimum function. However, average function is not associative so it may be harder to implement using standard algorithms and we will discuss the implementation later in this section.

\[ s_{a,j} = \sum_{i} p^{i}_{a,j} \]  

(16)
**Neighbor Selection** Once the neighbors are discovered and their indirect weights are computed, some of them may turn out to have a very low similarity with the source user. It is been observed in collaborative filtering systems that the neighbors with low correlations will greatly affect the prediction accuracy [17]. So in addition to the naive method that considers all the neighbors, we also consider two neighborhood selection techniques in our scheme.

- **Threshold Based Selection.** A threshold based neighborhood selection scheme sets an absolute similarity threshold, where all neighbors with absolute similarity greater than the threshold are selected. It has been suggested that high correlates can be exceptionally more valuable than those with lower correlations [17]. Hypothetically, setting a high threshold limits the neighborhood to containing very highly similar users, but may also results in a small or even empty neighborhood so it cannot compute the similarity based reputation for many users.

- **k Nearest Neighbor (kNN) selection** An alternative technique is to pick the top k neighbors in terms of similarity for each user. This works in a similar way to k nearest neighbor (kNN) classification algorithm.

Note that although the similarity based distance is defined in a symmetric way, i.e. the similarity between node $a$ and $b$ is the same as that between node $b$ and $a$, the neighborhood can be defined in an asymmetric way. Consider a scenario where node $a$ is connected to many nodes with high similarity and to node $b$ with low similarity while node $b$ is connected to a few nodes including node $a$ with low similarity. In this case, node $a$ may filter out node $b$ from its neighbor list by setting a fairly high threshold while node $b$ may very likely keep node $a$ as its neighbor because of its limited number of connected nodes. This notion of relative distance provides a nice notion of meta-distance into the neighborhood selection.

We will evaluate the different neighborhood selection techniques with varying parameters such as the threshold value in Section 3.4 in terms of their effects on sparsity resilience and robustness of the algorithms.

When a neighborhood selection scheme is used, the threshold or the $k$ value can be also
built into the neighborhood discovery process so that the neighbors who have a low weight will be pruned early on and the algorithm will be more efficient.

**Matrix Representation** If we represent the graph by a Boolean adjacency matrix $M$, $M^l$ will gives us all nodes that are reachable with a length $l$ path. If we represent the graph by a similarity weight matrix with each cell representing the link weight, assuming the weight is normalized for each user, we can also use matrix production $M^l$ to compute the indirect weight. It essentially corresponds to using production as the path concatenation function and using weighted average as the path combination function.

If we normalize the similarity weight by each user, we can also interpret the similarity inference problem as a stochastic transition process similar to the random walker model in PageRank [68] and EigenTrust [49]. Imagine the source user $a$ is trying to find similar neighbors, at each step, it hops to a neighbor according to the current users distribution of similarity. The process stops when $a$ reaches a user who has rating for $j$. User $a$ is more likely to be at similar users and takes his rating.

The difference of our approach from web of trust or EigenTrust is that user $a$ stops when it reaches the maximum path length. In the global inference approach such as EigenTrust, a users personal beliefs are washed out by the infinite aggregation of other users beliefs and the resulting EigenVector is a global trust vector.

It is worth mentioning, however, if we normalize the similarity weight by each user, then the similarity value does not have the semantic meaning any more, it only gives a relative rank of the users in terms of similarity. So the threshold based algorithm will not work with normalized similarity but $k$NN based scheme will.

**Implementation Algorithm**

We implement a breadth first search based algorithm of the above conceptual model in our first prototype.

Algorithm 7 depicts a sketch of the algorithm that is executed at node $a$. Node $a$ first initializes its neighbor by adding itself into the list. Then at iteration step $l$, it gets all the $l$-hop neighbors and compute their indirect weights. The algorithm terminates after it retrieves all the $L$-hop neighbors.
Algorithm 7 getNeighbors($a, L, f$)

Input: $a, L, f$, Output: $N_a$

$N_a.put((a,1))$

$l = 1$

while $l \leq l_{\text{max}}$ do

newNeighbors = $\phi$

for $b \in N_a$ do

$I_N_b = $ GetImmediateNeighbors($b$)

for $c \in I_N_b$ do

$w_{a,c} = w_{a,b} * w_{b,c}$

if $c \notin$ newNeighbors then

newNeighbors.put ($c, w_{a,c}$)

else

newNeighbors.update ($c, f$ (newNeighbors.get($c$), $w_{a,c}$))

end if

end for

end for

$N_a.add(\text{newNeighbors})$

$l = l + 1$

end while

At each iteration step, node $a$ go through its current neighbor list, and retrieve all the immediate neighbors of its current neighbors, if the node is not in us neighbors list yet, it is considered a new $l$-hop neighbor and being added into the newNeighbors list. If there are multiple paths from $a$ to $c$, it uses the given combination function (Maximum, Minimum or Average) to update its weight. Note that Average can be implemented by using Sum and Count and we skip the algorithm details here.

We can also adopt spreading activation algorithms for certain implementations of the above model to achieve better efficiency. Spreading activation models have been first introduced in order to simulate human comprehension through semantic memory and have been later applied to many associative retrieval problems [22]. The similarity inference problem can be naturally modeled as an association retrieval problem and various spreading activation algorithms can be used for implementing the computation. For example, the branch and bound algorithm [18] can be adopted to implement the above model if the path combination functions are associative (out of the functions we considered, maximum and minimum are associative, while average is not).
3.4 Experimental Evaluations

We performed a set of simulations to study the performance of the framework under different sparsity levels and threat models.

3.4.1 Experiment Setup

For each experiment, we vary the sparsity of the input feedback and evaluate the accuracy and coverage of different parameters to study the effect of sparse feedback.

Table 2: Countering Feedback Sparsity: Experiment Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community Setting</td>
<td>$N$ # of peers in the community</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>$s$ density of feedback</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>$k$ % of malicious peers in the community</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$mrate$ % of transactions a malicious peer acts malicious</td>
<td>100%</td>
</tr>
<tr>
<td>Trust Computation</td>
<td>$L$ Maximum Path Length</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Path combination function</td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td>$C_{r_{min}}$ Neighbor selection threshold</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$nExp$ # of experiments over which results are averaged</td>
<td>10</td>
</tr>
</tbody>
</table>

Based on the algorithmic scheme described in Section 3.3, we study the performance of the system experimentally with the following varying parameters.

- **Similarity measure.** We study the three similarity measures used to compute the direct similarity weight: pearson correlation, vector similarity, and Euclidean distance.

- **Maximum path length.** We vary the maximum path length that is used for exploring indirect neighbors from 1 to 4. When 1 is used, only direct neighbors are considered.

- **Path combination function.** We consider the three path combination functions used to compute the indirect similarity weight when there are multiple paths between two nodes: maximum, minimum, and average.

- **Threshold based neighbor selection.** We also vary the absolute threshold used for selecting a subset of neighbors to aggregate the feedback from 0 to 0.75. When 0 is
used, it is the same as using all the neighbors.

When one parameter is varied, we set the other parameters with a default value. Table 2 summarizes the main experimental parameters with their default values as well as the community parameters that we used for the experiments.

<table>
<thead>
<tr>
<th>Attack Models</th>
<th>Service Quality</th>
<th>Feedback Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Collusive</td>
<td>Low</td>
<td>Random</td>
</tr>
<tr>
<td>Collusive</td>
<td>Low</td>
<td>Target maximum rating (within group)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random (otherwise)</td>
</tr>
<tr>
<td>Collusive Camouflage</td>
<td>Low</td>
<td>Target maximum rating (within group)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Honest (otherwise)</td>
</tr>
</tbody>
</table>

We also perform attacks according to the attack strategies we described in Section 3.2.2 and evaluate the performance of the system with a low sparsity and varying sizes of attacks. Table 3 summarizes the behaviors of malicious peers in different attack models.

3.4.2 Evaluation Metrics

We examine different algorithmic variants of each component in the framework and evaluate their performance focusing on the sparsity resilience and robustness of the collaborative systems. We use the following two main metrics,

- **Accuracy.** An accuracy metric empirically measures how accurate a system in predicting the reputation scores as a measure of the trustworthiness of peers.

  We use statistical accuracy metrics that compare how reputation system’s computed reputation values for a peer differ from the peer’s assigned reliability value. Typical statistics metrics include Mean Absolute Error (MAE), Root Mean Squared Error and Correlation error. All the above metrics generally provided same conclusions, so we only report Root Mean Square Error.

- **Coverage.** Coverage measures the percentage of a test dataset that the reputation system is able to compute reputation value for. It is worth noting that coverage
and accuracy is at odds in some cases. For example, using neighborhood threshold selection may increase accuracy but decrease coverage. We will therefore focus on both improving accuracy and coverage in various sparsity levels and threat models.

3.4.3 Countering Sparsity

We first study how variant algorithmic components in our scheme handle the sparsity of the feedback.

Similarity measures. There are a number of ways to derive the similarity measure as the credibility weight for aggregating the feedback of peers. We first study their effects on the sparsity reputation systems.

![Error and Coverage Graphs](Image)

(a) Trust Computation Error  (b) Trust Computation Coverage

**Figure 9:** Trust Computation and Coverage of Different Similarity Measure with Varying Sparsity Levels

Figure 9 shows the trust computation error and coverage of using different similarity measures under different sparsity levels. The three similarity measures give comparable precision and coverage. In particular, Euclidean distance and vector similarity achieve better coverage than Pearson correlation as Pearson correlation can only be computed when two users have commonly rated peers.

Maximum Path Length. Now we study how the propagation of similarity affects the system in coping with sparsity. We first study the effect of the maximum path length that is used to explore for indirect neighbors.

Figure 10 shows the trust computation error and coverage of using different maximum path length under different sparsity levels. We can make a few interesting observations. First, the 2-Hop algorithms provide significantly better precision and coverage than the basic

63
Figure 10: Trust Computation Error and Coverage of Varying Maximum Path Length with Varying Sparsity Levels

1-Hop algorithm (maximum path length = 1). The performance gain is more significant when the input data is extremely sparse. This shows that considering indirect neighbors is important in countering sparse data. It reflects the intuition that you need to talk to people outside your close circle to get the best information. Second, 3-Hop and 4-Hop algorithms do not perform much better than 2-Hop algorithm. It shows that considering neighbors that are far away does not help much in achieving a better precision and coverage in this case. It also reiterates the observation that limited reputation sharing or a local reputation scheme is desired to avoid the unnecessary computation costs. This can be potentially explained by the small world effect wherein most of the peers may be connected by length 2 paths in our scenario. It is worth noting however that a formal analytical study is needed to determine the optimal maximum path length for a given sparsity and distribution of the community.

Path Combination Function. We also study the effect of different path combination functions when computing the similarity weight for indirect neighbors.

Figure 11 illustrates the trust computation error and coverage of the three path combination functions with varying sparsity levels. Interestingly, the maximum function has the lowest error rate with varying sparsity levels. This shows that by assuming an optimistic behavior (taking the maximum similarity value among multiple paths) peers can achieve better reputation accuracy given a sparse feedback input. Different path combination functions do not have any effect on the computation coverage.

Threshold Based Neighbor Selection. Finally we study the threshold based technique
in selecting neighbors for trust evaluations,

Figure 11: Trust Computation Error and Coverage of Different Path Combination Functions with Varying Sparsity Levels

Figure 12: Trust Computation Error and Coverage of Varying Neighbor Selection Threshold with Varying Sparsity Levels

Figure 12 illustrates the trust computation error and coverage of using different threshold values with varying sparsity levels. Interestingly, setting different threshold values does not have a significant impact on the trust precision. Having a threshold value of 0.75 increases the precision marginally but suffers a lower coverage.

3.4.4 Effect of Feedback Manipulation

Now we study how the scheme performs at a low sparsity with feedback manipulations. We consider each of the attack strategies we described earlier and study how different algorithm components perform under each of them.

Similarity Measure. We first study the performance of different similarity measures in terms of trust computation error and coverage under different threat models.
Figure 13: Trust Computation Error and Coverage of Different Similarity Measures in Non-Collusive Model

Figure 14: Trust Computation Error and Coverage of Different Similarity Measures in Collusive Model

Figure 13 shows the trust computation error and coverage using different similarity measures under non-collusive model where each individual malicious user acts independently. We see that the three similarity measures have the same effect on the computation error which increases as the percentage of malicious users increases. Pearson correlation and Euclidean distance give slightly better precision than vector similarity. This may be contributed to the fact that vector similarity assigns a default 0 value for the missing feedback rating when comparing two users which results in errors. The computation coverage is not affected.

For the collusive threat models, collusive and collusive camouflage, we assume the goal of the collusive malicious users is to boost their own ratings. In particular, in the collusive camouflage model, malicious users camouflage as honest users by providing honest ratings about the users outside their group in order to be similar to other users so that their dishonest (high) ratings about malicious users themselves can be counted. So in addition to
measure the trust computation errors for the whole population of users, we also measure the computation errors for the malicious users which will reflect how the collusive camouflage users achieve their malicious goal.

Figure 14 and 15 show the trust computation error and coverage using different similarity measures under collusive, and collusive camouflage threat model respectively. We observe that by colluding with each other, the malicious users are able to decrease the system computation accuracy to a large extent esp. when there is a majority of them in the system. The target error is even higher than the general error. Among the different similarity measures, pearson correlation is slightly more resistant to malicious users than the other two measures and vector similarity suffers the most. Another interesting phenomenon is that by using a collusive camouflage strategy, the malicious users are able to increase the target error to a larger extent than the collusive model, meaning their own ratings may be raised to
Figure 17: Trust Computation Error and Coverage of Varying Maximum Path Length in Collusive Model

Figure 18: Trust Computation Error and Coverage of Varying Maximum Path Length in Collusive Camouflage Model

a larger extent. However, the general error is decreased because the camouflaged malicious users provide honest ratings which actually make good contributions to the community feedback.

Maximum Path Length. We next study the effect of neighbors’ maximum path length on the trust computation error and coverage with a sparse feedback under different threat models.

Figure 16 shows the trust computation error and coverage under non-collusive model. We can again make a few interesting observations. First, the 2-Hop algorithms and above provide significantly better precision and coverage than the basic 1-Hop algorithm. This shows that considering indirect neighbors is important in countering sparse data and as a result it also provides a better resilience to the manipulation of feedback by malicious users. Second, 3-Hop and 4-Hop algorithms do not perform much better than 2-Hop algorithm.
This shows that considering neighbors that are far away does not help much in coping with malicious feedback manipulation.

Figure 17 and 18 show the trust computation error and coverage results under collusive and collusive camouflage threat model respectively. We see a similar trend as in the non-collusive model but to a larger extent. Also, the collusive camouflage model is able to slightly increase the trust computation error for the malicious group.

Path Combination Function. We next study the effect of different path combination functions in each of the threat models.

Figure 19 shows the trust computation error and coverage using different path combination functions under non-collusive model. The three path combination functions have a similar effect on the trust evaluation error which increases as the percentage of malicious users increases. The minimum function performs slightly worse than the other two similar to what we have observed in their effects on countering sparsity.
Figure 21: Trust Computation Error and Coverage of Different Path Combination Functions in Collusive Camouflage Model

Figure 22: Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Non-Collusive Model

Figure 20 and 21 show the trust computation error and coverage using different path combination functions under collusive and collusive camouflage threat model respectively. Again, the graphs show a similar trend as in the non-collusive model but to a larger extent.

**Threshold-Based Neighbor Selection.** Finally we study how threshold-based neighbor selection help countering feedback manipulation attacks.

Figure 22, 23, and 24 show the trust computation error and coverage using different neighbor selection threshold under non-collusive, collusive, and collusive camouflage threat model respectively. Again, the graphs show a similar trend as in the non-collusive model but to a larger extent. We observe that when we increase the threshold, the precision of the algorithm increases as it helps to filter out malicious peers when aggregating opinions. However, as we have expected, a higher threshold also decreases the coverage of the reputation computation.
Figure 23: Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Collusive Model

Figure 24: Trust Computation Error and Coverage of Varying Neighbor Selection Threshold in Collusive Camouflage Model

3.5 Summary

We presented in this chapter a similarity inference scheme that helps the system coping with feedback sparsity with potential feedback manipulations. We studied various algorithmic components in the scheme including the similarity measure and the similarity inference model. Our experimental evaluations show that by considering indirect neighbors, the scheme provides resilience for the trust framework against feedback sparsity even with various feedback manipulation attacks.
CHAPTER IV

COUNTERING FEEDBACK OSCILLATION

We have presented the basic trust framework with the enhancement to cope with feedback sparsity and vulnerabilities. As most existing reputation systems, we have computed the reputation value by aggregating the feedback of a node over the recent period using an average. A malicious node may strategically alter its behavior in a way that benefits itself. For example, it may behave non-maliciously until it attains a good reputation (reflected in its trust value) and then behave maliciously. Or it could oscillate between building and milking reputation. A dependable system should be capable of handling or penalizing the dynamic and strategic changes in the behavior of malicious nodes. For example, a node’s reputation (represented by its trust value) must drop quickly as soon as it misbehaves, while it should be hard to increase within a short period of time so a malicious node will not gain or even penalized by strategically altering its behavior. In addition, the trust building techniques based on reputation should be robust in the presence of sudden and possibly unpredictable changes in the behavior of malicious nodes.

In this chapter, we present a strategic oscillation guard built on top of the feedback aggregation to effectively handle such strategic behaviors by malicious nodes. In addition to the current reputation that is computed from aggregating feedback over the recent period, we promote the incorporation of the reputation history and behavior fluctuations of nodes into the estimation of their trustworthiness. We use adaptive parameters to allow different weighting functions to be applied to current reputation, reputation history, and reputation fluctuations. Several optimization techniques are developed to ensure the effectiveness and practical applicability of the proposed solution. For instance, we introduce the concept of fading memories to reduce the cost of maintaining historical information of nodes.
4.1 Threat Model and Design Goals

We define a strategic malicious node as a node that adapts its behavioral pattern (with time) so as to maximize its malicious goals. Consider a scenario wherein a bad node does not misbehave until it earns a high trust value. The scenario becomes more complicated when bad nodes decide to alternate between good and bad behavior at regular or arbitrary frequencies. In this chapter, we primarily focus on strategic oscillations by malicious nodes and describe concrete and systematic techniques to address both steady and sudden changes in the behavioral pattern of a node without adding heavy overheads to the system. Other possible behavioral strategies that could be employed by malicious nodes are not considered in this chapter.

A dependable trust model should be capable of handling the following four important issues: (P1) sudden fluctuations in node behavior, (P2) distinguish an increase and decrease in node behavior, (P3) tolerate unintentional errors, and (P4) reflect consistent node behavior. We propose a dependable trust model that computes reputation-based trust of a node by taking into consideration: current feedback reports about the node, its historical reputation, and the fluctuations in the node’s current behavior.

We present an optimization theory based cost metric to formalize the above design goals.

The primary goal of our safeguard techniques is to maximize the cost that the malicious nodes have to pay in order to gain advantage of the trust system. As we have briefly discussed in Chapter 2, we first formally define the behavior of a non-malicious and a malicious node in the system using the game theory approach [27]. A non-malicious node is the commitment type and a long-run player who would consistently behave well, because cooperation is the action that maximizes the player’s lifetime payoffs. In contrast a strategic malicious node corresponds to an opportunistic player who cheats whenever it is advantageous for him to do so. Now we formally describe a cost model for building reputation-based trust and use this cost model to illustrate the basic ideas of maximizing the cost (penalty) to be paid by anyone behaving maliciously. Let $TV_n(t)$ denote the trust value as evaluated by the system for node $n$ at time $t$ ($0 \leq TV_n(t) \leq 1$). Let $BH_n(t)$ denote the actual behavior of node $n$ at time $t$ ($0 \leq BH_n(t) \leq 1$), modeled as the fraction of transactions that would be
honestly executed by node $n$ between an infinitesimally small time interval $t$ and $t + dt$. We define the extent to which a malicious node may misuse its reputation (denoted by $X_n(t)$) and the amount of work to be done at time $t$ by a malicious node in order to increase its reputation-based trust value (denoted by $Y_n(t)$) as $X_n(t) = \max(TV_n(t) - BH_n(t), 0)$ and $Y_n(t) = \max(BH_n(t)TV_n(t), 0)$ respectively. Then, we define the cost function for a node $b$ as shown in Equation 17.

$$\text{cost}(b) = \lim_{t \to \infty} \frac{1}{t} \int_0^t (BH_b(x) - TV_b(x))dx$$

(17)

Let $G$ be the set of good nodes and $B$ be the set of bad nodes. The objective is $\forall g \in G : TV_g(t) \approx 1$ and $\forall b \in B : \text{cost}(b)$ is maximized. Figure 25 provides an intuitive illustration of the above cost function for a strategic malicious node oscillating between acting good and bad. Referring to Figure 25, observe that the problem of maximizing the cost paid by the malicious nodes can be reduced to maximizing the area under $Y_n(t) - X_n(t)$, that is, minimizing the extent of misuse ($X_n(t) = \max(TV_n(t) - BH_n(t), 0)$) and maximizing the cost of building reputation ($Y_n(t) = \max(BH_n(t) - TV_n(t), 0)$).

![Figure 25: Cost of Building Reputation](image)

In addition to maximizing the cost metric, our goal is to ensure that any node behaving well for an extended period of time attains a good reputation. However, we should ensure that the cost of increasing a node’s reputation depends on the extent to which the node misbehaved in the past. For example a node that misbehaved for an extended period of time in the past should find it very hard to build reputation in a short span of time (although it is eventually possible).

## 4.2 Dependable Trust Model

Bearing the above analysis in mind, we present a dependable trust model in this section.
4.2.1 Trust Computation Model

Let $R(t)$ denote the raw trust value of node $n$ at time $t$. The simplest form to calculate $R(t)$ can be an average of the ratings over the recent period of time. Let $TV(t)$ denote the dependable trust value of node $n$ at time $t$ and we compute $TV(t)$ using Equation 18. Note that $R'(t)$ denotes the derivative of $R(x)$ at $x = t$.

$$TV(t) = \alpha \times R(t) + \beta \times \frac{1}{t} \int_0^t R(x)dx + \gamma \times R'(t)$$  \hspace{1cm} (18)

Equation 18 resembles a Proportional-Integral-Derivative (PID) controller used in control systems [67]. The first component (proportional) refers to the contribution of the current reports received at time $t$. The second component (integral) represents the past performance of the node (history information). The third component (derivative) reflects the sudden changes in the trust value of a node in the very recent past. Choosing a larger value for $\alpha$ biases the trust value of a node $n$ to the reports currently received about $n$. A larger value of $\beta$ gives heavier weight to the performance of the node $n$ in the past. The averaging nature of the proportional and integral components enables our model to tolerate errors in raw trust values $R_n(t)$ ($P3$) and reflect consistent node behavior ($P4$). A larger value of $\gamma$ amplifies sudden changes in behavior of the node in the recent past (as indicated by the derivative of the trust value) and handles sudden fluctuations in node behavior ($P1$). We discuss techniques to distinguish increase and decrease in node behavior ($P2$) later in this Section.

![Diagram](image)

**Figure 26:** PID Based Dependable Trust Model: Illustration
4.2.2 Realizing the Trust Model

We now describe a simple implementation of the dependable trust model described above.
For simplicity, we assume that the trust values of nodes are updated periodically within
each time period $T$. Let successive time periods (intervals) be numbered with consecutive
integers starting from zero. We call $TV[i]$ the dependable trust value of node $n$ in the
interval $i$. $TV[i]$ can be viewed as a function of three parameters: (1) the feedback reports
received in the interval $i$, (2) the integral over the set of the past trust values of node $n$,
and (3) the current derivative of the trust value of node $n$.

Incorporating feedbacks by computing $R[i]$. Let $R[i]$ denote the raw reputation value
of node $n$ computed as an aggregation of the feedbacks received by node $n$ in interval $i$.
$R[i]$ can be computed by using a simple average over all the feedback ratings received by
node $n$ in time interval $i$.

Incorporating History by Computing Integral. We now compute the integral (history)
component of the trust value of node $n$ at interval $i$, denoted as $H[i]$. Suppose the
system stores the trust value of node $n$ over the last $maxH$ (maximum history) intervals,$H[i]$ could be derived as a weighted sum over the last $maxH$ reputation values of node $n$
using Equation 19.

$$H[i] = \sum_{k=1}^{maxH} R[i - k] * \frac{w_k}{\sum_{k=1}^{maxH} w_k}$$

(19)

The weights $w_k$ could be chosen either optimistically or pessimistically. An example of an
optimistic summarization is the exponentially weighted sum, that is, $w_k = \rho^{k-1}$ (typically,
$\rho < 1$). Note that choosing $\rho = 1$ is equivalent to $H$ being the average of the past
$maxH$ reputation values of node $n$. Also, with $\rho < 1$, $H$ gives more importance to the more recent
reputation values of node $n$. We consider these evaluations of $H$ optimistic since they allow
nodes to attain higher trust values rather quickly. On the contrary, a pessimistic estimate
of $H$ could be obtained with $w_k = \frac{1}{\ln(1 - \rho)}$. Such an evaluation assigns more importance to
those intervals where the node behaved particularly badly.

Strengthening the dependability of $TV[i]$. Once we have calculated the feedback-based
reputation ($R[i]$) for the node $n$ in the interval $i$ and its past reputation history ($H[i]$), we
can use Equation 20 to compute the derivative component \((D[i])\). Note that Equation 20 uses \(H[i]\) instead of \(R[i-1]\) for stability reasons.

\[
D[i] = R[i] - H[i]
\]

(20)

We now compute the dependable trust value \(TV[i]\) for node \(n\) in the interval \(i\) using Equation 21:

\[
TV[i] = \alpha \cdot R[i] + \beta \cdot H[i] + \gamma(D[i]) \cdot D[i]
\]

(21)

where

\[
\gamma(x) = \begin{cases} 
\gamma_1 & \text{if } x \geq 0 \\
\gamma_2 & \text{if } x < 0 
\end{cases}
\]

(22)

In this equation, \(TV[i]\) is derived by associating different weights \(\gamma_1\) and \(\gamma_2\) for a positive gradient and a negative gradient of the trust value respectively, enhancing the dependability of \(TV[i]\) with respect to sudden behavioral changes of node \(n\). One of the main motivations in doing so is to set \(\gamma_1 < \beta < \gamma_2\), thereby increasing the strength of the derivative component (with respect to the integral component) when a node shows a fast degradation of its behavior, and lowering the strength of the derivative component when a node is building up its reputation (recall \(P2\) in our design goal). Our experiments (see Section 4.3) show that one can use the rich set of tunable parameters provided by Equation 21 to handle both steady and sudden changes in the behavior of a strategic malicious node.

4.2.3 Optimization through Fading Memories

We have proposed to compute the dependable trust value of a node \(n\) in interval \(i\) based on its current reputation, its reputation history prior to interval \(i\) and its reputation fluctuation. In computing reputation history, we assume that the system stores the reputation-based trust values of node \(n\) for the past \(maxH\) number of intervals. By using a smaller value for \(maxH\), we potentially let the wrong-doings by a malicious node to be forgotten in
approximately $maxH$ time intervals. However, using a very large value for $maxH$ may not be a feasible solution for at least two reasons: (i) The number of trust values held on behalf of a long standing member of the system could become extremely large. (ii) The computation time for our trust model (Equations 19 and 21) increases with the amount of data to be processed. In the first prototype, we introduce fading memories as a performance optimization technique to reduce the space and time complexity of computing $TV[i]$ by allowing a trade-off between the history size and the precision of the historical reputation estimate.

One simple technique to trade-off history size with precision would be to aggregate the trust value in each $k$ consecutive intervals into one value. However, from our experiments we observed that it is vital to keep the trust values in the recent past very precise. Therefore, we propose to aggregate data over intervals of exponentially increasing length in the past $\{k^0, k^1, \ldots, k^{m-1}\}$ into $m$ values (for some integer $k > 0$). Observe that the aggregates in the recent past are taken over a smaller number of intervals and are hence more precise. This permits the system to maintain more detailed information about the recent trust values of node $n$ and retain fading memories (less detailed) about the older trust values of node $n$. Given a fixed value to the system-defined parameter $m$, one can trade-off the precision and the history size by adjusting the value of $k$.

![Figure 27: Updating Fading Memories: Illustration](image)

Now we describe how we implement fading memories. To simplify the discussion, let us assume that $k = 2$. With fading memory optimization, our goal is to summarize the last $2^m - 1$ $(\sum_{i=0}^{m-1} 2^i = 2^m - 1)$ trust values of a node by maintaining just $m$ $(=\log_2(2^m))$
values.

This can be done in two steps. (i) we need a mechanism to aggregate $2^m - 1$ trust values into $m$ values, and (ii) we need a mechanism to update these $m$ values after each interval.

We perform Step 1 as follows. In the interval $t$, the system maintains trust values in intervals $t - 1, t - 2, \cdots, t - 2^m$ in the form of $m$ trust values by summarizing intervals $t - 2^j, t - 2^j - 1, \cdots, t - 2^{j+1} + 1$ for every $j (j = 0, 1, \cdots, m - 1)$, instead of maintaining one trust value for each of the $2^m - 1$ time intervals. Figure 27 provides an example where $k = 2$ and $m = 3$. $FTV[i]$ denotes the faded values at time $t$ and $FTV'[i]$ denotes the faded values at time $t + 1$.

Now we discuss how we perform Step 2. Let $FTV'[j] (0 \leq j \leq m - 1)$ denote the faded trust values of node $n$ at interval $t$. Ideally, re-computing $FTV$ for interval $t$ requires all of the past $2^m - 1$ trust values. With fading memories we only store $m$ summarization values instead of all the $2^m - 1$ trust values. Thus, at interval $t$ we approximate the trust value for an interval $t - i (1 \leq i \leq 2^m)$ by $FTV[i] \lceil \log_2 i \rceil$. We use Equation 23 to approximate the updates to the faded trust values for interval $j (j = 0, 1, 2, \cdots, m - 1)$ with the base case $FTV[0] = R[t]$. Figure 27 gives a graphical illustration of Equation 23 for $m = 3$.

$$FTV'[j] = \frac{(FTV'[j] \times (2^j - 1) + FTV'[j - 1])}{2^j}$$

(23)

In summary, the fading memories approach closely resembles the way human beings remember their experiences. Our experiments show the vital role of fading memories in optimizing the performance of the trust system while maintaining its dependability at trust model level.

4.3 Experimental Evaluation

In this section, we report results from our simulation based experiments to evaluate our approach to handle dynamic node behaviors.

We first study the behavior of our guard against strategic oscillations by comparing the optimistic and pessimistic summarization techniques. We demonstrate the significance of various parameters in our dependable trust metrics by varying the weights assigned to reports received in the recent time window ($\alpha$), the history ($\beta$), and the derivative
component ($\gamma$). Then, we show the impact of history size ($maxH$) on the effectiveness of our trust model and the advantages of storing past experiences using fading memories.

### 4.3.1 Experiment Setup

We implemented our simulator using a discrete event simulation [32] model. Our system comprises of about $N = 1024$ nodes; a random $p\%$ of them is chosen to behave maliciously. In the following portions of this section, we demonstrate the effectiveness of the PID model for coping with strategic behaviors of malicious nodes.

![Node Behavior](image)

**Figure 28:** Oscillating Behavior Model I

For all experiments reported in this section, we studied four different models of strategic malicious behaviors. In Model I shown in Figure 28, the malicious nodes oscillate from good to bad behavior at intervals of regular time periods. In model II, the malicious nodes oscillate between good and bad behaviors at exponentially distributed intervals. In model III, the malicious nodes choose a random level of goodness and stay that level for an exponentially distributed duration of time. In model IV the malicious node shows a sinusoidal change in its behavior that is the node steadily and continuously changes its behavior unlike models I, II and III which show sudden fluctuations.

### 4.3.2 Comparing Optimistic and Pessimistic Summarizations

We first compare the two types of weighted summarization techniques discussed in section 4.2.2. Figure 29 shows the values obtained on summarization given the node behavior model I shown in Figure 28 (using $\rho = 0.7$, $maxH = 10$ and time period of malicious behavior oscillation = 10). The result shows that mean value ($mean$) and exponentially weighted sum ($exp$) have similar effect and they both are more optimistic than the inverse trust
value weighted sum (invtv). Observe that the more pessimistic a summarization is, the harder it is for a node (both malicious and non-malicious) to attain a high trust value in a short span of time and the easier it is to drop its trust value very quickly (recall our cost function in Equation 17). Also observe that the exponentially weighted sum in comparison to the mean rises quite steeply making it unsuitable for summarization.

4.3.3 Trust Model Parameters

Figure 30: Effect of Varying Parameters in the PID Trust Model

Figure 30 shows the results obtained from our trust model with various parameter settings under the malicious behavior shown in model I (m1). alpha shows the results obtained when \( \alpha \) is the dominant parameter (\( \alpha \gg \beta, \gamma \)). With a dominant \( \alpha \) the trust model almost follows the actual behavior of the node since it amounts to disregarding the history or the current fluctuations in the behavior of the node (see Equation 18). beta-invtv shows the results obtained with \( \beta \) as the dominant parameter using inverse trust value weighted sum. With more importance given to the behavior history of a node, the trust value of a node does not change very quickly. Instead it slowly and steadily adapts to its actual behavior. gamma shows the results obtained with \( \gamma \) being the dominant factor. With
a large $\gamma$ the trust value responds very swiftly to sudden changes in the behavior of the node. Observe the steep jumps in the trust value that correspond to the time instants when the node changes its behavior. These results match our intuition, namely, $\alpha$, $\beta$ and $\gamma$ are indeed the weights attached to the current behavior, historical behavior and the fluctuations in a node’s behavior. Finally, non-adaptive shows the trust value of a node in the absence of dependable schemes to handle dynamic node behaviors. From Figure 30 it is evident that the cost paid by a malicious node in a non-adaptive model is almost zero, while that in a dependable model is quite significant. A more concrete evaluation that considers the combined effect of various trust model parameters is a part of our ongoing work.

4.3.4 Varying History Size

![PID Trust Model with Varying History Size](image)

**Figure 31**: PID Trust Model with Varying History Size

In this section we show the effect of history size $maxH$ on the cost (see Equation 17) paid by malicious nodes. Figure 31(a) shows a scenario wherein the malicious nodes oscillate in their behavior every 10 time units, Note that in this experiment we used $\alpha = 0.2$, $\beta = 0.8$, $\gamma_1 = 0.05$ and $\gamma_2 = 0.2$. Based on our experiences with the dependable trust model one needs to choose $\alpha$ and $\beta$ such that $\frac{\beta}{\alpha}$ is comparable to $maxH$ (intuitively, this weights the history component in proportion to its size ($maxH$)). Note that this experiment uses $maxH = 5$ which is less than the time period of oscillations by the malicious nodes. From Figure 31(a) it is clear that the dependable trust models (TrustGuard-adaptive in figure) performs better in terms of cost to be paid by the malicious nodes than the non-adaptive trust model (recall the cost model in Section 4.1). However, this does not entirely maximize the cost paid by malicious nodes. Figure 31(b) shows the trust values obtained when $\alpha = 0.1$, $\beta = 0.9$, $\gamma_1 = 0.05$ and $\gamma_2 = 0.2$. These results match our intuition, namely, $\alpha$, $\beta$ and $\gamma$ are indeed the weights attached to the current behavior, historical behavior and the fluctuations in a node’s behavior. Finally, non-adaptive shows the trust value of a node in the absence of dependable schemes to handle dynamic node behaviors. From Figure 30 it is evident that the cost paid by a malicious node in a non-adaptive model is almost zero, while that in a dependable model is quite significant. A more concrete evaluation that considers the combined effect of various trust model parameters is a part of our ongoing work.
\( \gamma_1 = 0.05, \gamma_2 = 0.2 \) and \( \text{max}H = 15 \) (larger than the time period of oscillation by the malicious node). Clearly, having a larger history ensures that one can maximize the cost paid by the malicious nodes. In fact, one observes that the cost paid by malicious nodes for \( \text{max}H \) equal to 5, 10 and 15 are in the ratio of 0.63 : 1 : 3.02 respectively. This observation tells us that if a strategic malicious node knew that \( \text{max}H = 5 \), then it would oscillate at a period equal to 5 time intervals since anyway the system does not remember its past performance beyond 5 time intervals. In short, by knowing the exact value of \( \text{max}H \), a strategic malicious node would start to oscillate with time period equal to \( \text{max}H \) so as to minimize its cost. It is interesting to note that, when the non-adaptive model is used, the cost paid by malicious nodes is close to zero for all values of time period of behavior oscillation and history size \( \text{max}H \).

4.3.5 Fading Memories

![Figure 32: PID Trust Model with Fading Memories](image)

We now evaluate the effectiveness of the fading memories technique in efficiently storing the performance of a node over the last \( 2^{\text{max}H} \) intervals using a logarithmically small number of values. Figure 32 shows the effect of using fading memories when a malicious node oscillates with time period equal to 100 time units. It compares a dependable trust model (\( \text{TrustGuard-adaptive} \) in figure) with \( \text{max}H = 10 \) and a dependable trust model using fading memories (\( \text{TrustGuard-ftv} \) in figure) based technique with \( m = 8 \). From Figure 32 it is apparent that using a simple adaptive technique with \( \text{max}H = 10 \) enables a bad node to recover from its bad behavior that stretched over 100 time units in just 10 additional time units, since the past performance of the node is simply forgotten after 10 time units. As we discussed in section 4.1, one of the design principles for dependable trust management is to
prevent a bad node that has performed poorly over an extended period of time to attain a high trust value quickly. Clearly, the adaptive fading memories based technique can perform really well in this regard, since using just 8 values, it can record the performance of the node over its last 256 ($2^8$) time intervals. It is important to note that the solution based on fading memories has bounded effectiveness in the sense that by setting $m = 8$, any node could erase its malicious past over 256 time intervals. However, the key benefit of our fading memories based approach is its ability to increase the cost paid by malicious nodes, with minimal overhead on the system performance.

4.3.6 Other Strategic Oscillation Models

We also studied the cost of building reputation under different bad node behavior models discussed in the beginning of Section 4.3.1. From our experiments, we observed that the response of our trust model towards models II, III and IV are functionally identical to that obtained from model I (Figure 28). However, from an adversarial point of view, we observed that these strategies do not aid in minimizing the cost to be paid by malicious nodes to gain a good reputation when compared to model I. In fact, the cost paid by malicious nodes using models I, II, III and IV are in the ratio of $1 : 2.28 : 2.08 : 1.36$. In models II and III, the malicious nodes do not pursue their malicious activities the very moment they attain a high reputation. In model IV, the malicious nodes slowly degrade their behavior, which does not given them good benefits (see the extent of misuse $X_n(t)$ in Figure 25) when compared to a steep/sudden fall. Hence, a strategic malicious node that is aware of $maxH$ would oscillate with time period $maxH$ in order to minimize its cost (refer Equation 17). Nonetheless this emphasizes the goodness of our dependable trust model since it is capable of effectively handling even its worst vulnerability (model I with oscillation time period $maxH$).

4.4 Summary

We have presented models and techniques as countermeasures against strategic behaviors of malicious nodes. We proposed to measure the trustworthiness of peers based on current reputation, reputation history and reputation fluctuation and develop formal techniques to counter strategic oscillation of malicious nodes. We have demonstrated the effectiveness
of these techniques through an extensive set of simulation based experiments. We believe that the approach can efficiently and effectively guard a large-scale distributed reputation system, making it more dependable than other reputation systems.
CHAPTER V

PRESERVING FEEDBACK PRIVACY: PRIMITIVE PROTOCOLS

We have described the trust framework and a set of defense mechanisms that make the framework dependable against various vulnerabilities. Through the computation, we have assumed that peers are willing to share their ratings about others. After all, that is the assumption underneath a reputation system that harnesses the community wisdom. However, reputation aggregation also raises privacy concerns as participants may not wish to share their feedback with others for various reasons. For example, a user who has a low or negative rating about another user may not be willing to share it with the fear of being retaliated by the other user in the future. It is been observed in eBay that most feedback are positive because users are reluctant to file negative feedback for this reason. This motivates the importance of preserving feedback privacy in a reputation system. The main research challenge is how to share the reputation data while respecting the privacy constraints of the data owners. Interestingly, this is also a general problem of data sharing and integration in open communities. Because of the open and untrusted environments, individual peers or databases are not willing to fully disclose their private data.

The next two chapters address the problem of loss of privacy in the reputation system and the challenge of sharing data while respecting privacy constraints of the individual parties.

We propose a model in which users control their feedback data and protocols whereby a set of users can compute a public aggregation of their feedback data that does not expose individual users. We tackle the problem by first developing a suite of decentralized privacy-conscious computation protocols for important primitive operations among multiple private databases (such as MAX, MIN, SUM, and TopK). We adopt the paradigm of information integration with minimal necessary sharing [9] for developing privacy-preserving...
computation protocols. As a tradeoff for efficiency and practicability, the constraint of not revealing any additional information apart from the final result can be relaxed sometimes to allow minimal additional information to be revealed. In particular, we propose a novel probabilistic protocol for computing MAX, MIN and TopK functions that on one hand effectively minimize the information disclosure of individual databases and on the other hand are efficient in terms of both computation and communication costs. We show next how aggregate functions can be reduced or divided into individual steps that utilize the above protocols for individual functions. In particular, we model a set of reputation computation mechanism as $k$NN classification problem and propose a model for privacy preserving $k$NN classification. We illustrate how the model can be implemented with the top$k$ protocol proposed above and an existing summation protocol.

In this chapter, we present a set of protocols we developed for data integration functions in a general context. In next chapter, we model a set of reputation computation algorithms we have proposed in earlier chapters as $k$NN classification problem and present a privacy preserving $k$NN classification framework.

### 5.1 Problem Statement

Information integration has long been an important area of research as there is great benefit for organizations and individuals in sharing their data. Traditionally, information integration research has assumed that information in each database can be freely shared. Recently, it has been recognized that concerns about data privacy increasingly becomes an important aspects of the data integration because organizations or individuals do not want to reveal their private databases for various legal and commercial reasons.

**Application Scenarios.** The increasing need for privacy preserving data integration is driven by several trends [9]. In the business world, with the push of end-to-end integration between organizations and their suppliers, service providers, and trade partners, information sharing may occur across multiple autonomous enterprises. Full disclosure of each database is undesirable. It is also becoming common for enterprises to collaborate in certain areas and compete in others. This in turn requires selective information sharing.
Another important application scenario is driven by security. Government agencies realize the importance of sharing information for devising effective security measures. For example, multiple agencies may need to share their criminal record databases in identifying certain suspects under the circumstance of a terrorist attack. However, they cannot indiscriminately open up their databases to all other agencies.

In reputation systems, reputation aggregation also raises privacy concerns as participants may not wish to share their feedback with others for various reasons. For example, a user who has a low rating about another user may not be willing to share it being afraid of retaliations. A research challenge is how to share the reputation data while respecting the privacy constraints of the data owners.

Such concerns of data privacy place limits on the information integration. We are faced with the challenge of data integration while respecting privacy constraints. Ideally, given a database query spanning multiple private databases, we wish to compute the answer to the query without revealing any additional information of each individual database apart from the query result.

**Current Techniques and Research Challenges.** There are two main existing techniques that one might use for building the privacy preserving data integration applications and we discuss below why they are inadequate.

One technique is to use a trusted third party and have the participating parties report the data to the trusted third party, which performs the data integration task and reports back the result to each party. However, finding such a trusted third party is not always feasible. The level of trust required for the third party with respect to intent and competence against security breaches is too high. Compromise of the server by hackers could lead to a complete privacy loss for all participating parties should the data be revealed publicly.

The other is the secure multi-party computation approach [36, 35] that developed theoretical methods for securely computing functions over private information such that parties only know the result of the function and nothing else. However, the methods require substantial computation and communication costs and are impractical for multi-party large database problems.
Agrawal et al [9] recently proposed a new paradigm of information integration with minimal necessary sharing across private database. As a tradeoff for efficiency and practicability, the constraint of not revealing any additional information apart from the query result can be relaxed sometimes to allow minimal additional information to be revealed. As an example, they developed protocols for computing intersection and equijoin between two parties that is still based on cryptographic primitives but more efficient with minimal information disclosure.

Given this paradigm, research opportunities arise for developing efficient specialized protocols for different operations. One important operation is statistics queries over multiple private databases, such as top\(k\) data values of a sensitive attribute. In particular, when \(k = 1\), it becomes the max(min) query. For example, a group of competing retail companies in the same market sector may wish to find out statistics about their sales, such as the top sales revenue among them, but to keep the sales data private at the same time. The design goal for such protocols is two fold. First, it should be efficient in terms of both computation and communication costs. In order to minimize the computation cost, expensive cryptographic operations should be limited or avoided. Second, it should minimize the information disclosure apart from the query results for each participant.

**Organizations.** Bearing these design goals in mind, we propose a protocol for selecting top\(k\) data values of a sensitive attribute across multiple \((n > 2)\) private databases. The chapter has a number of unique contributions. First, we formalize the data privacy goal and the notion of loss of privacy in terms of information revealed by proposing a data privacy metric (Section 5.2). Second, we propose a novel probabilistic decentralized protocol for privacy preserving top\(k\) selection (Section 5.3). Third, We perform a formal analysis of the protocol in terms of its correctness, efficiency and privacy characteristics (Section 5.4) and evaluate the protocol experimentally (Section 5.5). We conclude the chapter with a brief summary (Section 5.6).
5.2 Problem Statement

In this section we define the problem of top\(k\) queries across private databases. We present the privacy goal that we focus in the chapter, followed by privacy metrics for characterizing and evaluating how the privacy goal is achieved.

The input of the problem is a set of private databases. A top\(k\) query is to find out the top\(k\) values of a common attribute of all the individual databases. We assume all data values of the attribute belong to a publicly known data domain. Now the problem is to select the top\(k\) values with minimal disclosure of the data values each database has besides the final result.

5.2.1 Adversary Model

We adopt the semi-honest model [35] that is commonly used in multi-party secure computation research for privacy adversaries. A semi-honest party follows the rules of the protocol, but it can later use what it sees during execution of the protocol to compromise other parties' data privacy. Such kind of behavior is referred to as honest-but-curious behavior [35] and also referred to as passive logging [91] in research on anonymous communication protocols.

The semi-honest model is realistic for our context based on the following observation. Today multiple organizations in the same market sectors are actively competing as well as collaborating with constantly evolving alliances. These parties often wish to find out aggregation statistics of their sales, such as the total sales or the top\(k\) sales among them in a given category or time period, while keeping their own sales data private. As a result, each participating party will want to follow the agreed protocol to get the correct result for their mutual benefits and at the same time reduce the probability and the amount of information leak (disclosure) about their private data during the protocol execution due to competition or other purposes.

Other adversary models include malicious model where an adversary can misbehave in arbitrary ways. In particular, it can change its input before entering the protocol or even terminates it arbitrarily. Possible attacks under this model include spoofing attack and
hiding attack where an adversary sends a spoofed dataset or deliberately hides all or part of its dataset and leads to a polluted query result. We plan to study the malicious model in our future work.

5.2.2 Privacy Goal

We focus on the data privacy goal for topk queries in this chapter. Ideally, besides the final topk results that are public to all the databases, nodes should not gain any more information about each other's data. As we have discussed earlier, with a centralized third party approach, all participating organizations will have to trust this third party and disclose their private data to the third party, which is not only costly in terms of legal and administration procedure but also undesirable by many. We propose a decentralized approach without any third trusted party. Our goal is to minimize data exposure among the multi-parties apart from the final result of the topk query.

We describe the different types of data exposure we consider and discuss our ultimate privacy goal in terms of such exposures. Given a node i and a data value \( v_i \) it holds, we identify the following data exposures in terms of the level of knowledge an adversary can deduce about \( v_i \): (1) Data value exposure: an adversary can prove the exact value of \( v_i \) (\( v_i = a \)), (2) Data range exposure: an adversary can prove the range of \( v_i \) (\( a \leq v_i \leq b \)) even though it may not prove its exact value, and (3) Data probability distribution exposure: an adversary can prove the probability distribution of \( v_i \) (\( pdf(v_i) = f \)) even though it may prove neither its range nor exact value.

Both data value and data range exposures can be expressed by data probability distribution exposure, in other words, they are special cases of probability distribution exposure. Data value exposure is again a special case of data range exposure. Intuitively, data value exposure is the most detrimental privacy breach. Due to the space restriction, we will focus our privacy analysis on the data value exposures in the rest of this chapter.

Similar to the exposures at individual node, we can consider data exposures from the perspective of a group of nodes by treating this subset of nodes as an entity. Note that even if a group's privacy is breached, an individual node may still maintain its privacy to
some extent. For example, an adversary may be able to prove that a group of nodes has a certain value but it is not certain which exact node has the value. In other words, the $m$-anonymity [85] is preserved given the size $m$ of the group.

The privacy goal we aim at achieving is to minimize the degree of data value exposures for each individual node. This includes the principle that we are treating all the nodes in the system equally and no extra considerations will be given to the nodes who contribute to the final top$k$ values (e.g., the node who owns the global maximum value). In addition to protecting the data exposure of each node, a related goal could be protecting the anonymity of the nodes who contribute to the final results, though it is not the focus of this chapter.

5.2.3 Privacy Metrics

Given the data privacy goal, we need to characterize the degree with which the privacy is attained. The key question is how to measure the amount of disclosure during the computation and what privacy metrics are effective for such measurement. Concretely, we need to quantify the degree of data exposure for a single data item $v_i$ that node $i$ holds. Let us first consider an existing metric and discuss why it is inadequate. We then propose a general and improved metric for data privacy.

The metric that one might use is the probabilistic privacy spectrum [72] proposed for web transactions anonymity and was also adopted for document ownership privacy later [12]. Now we need to evaluate whether we can adapt it for our data privacy purpose. Assuming an adversary is able to make a claim $C$ about the data value $v_i$ (e.g., $v_i = a$), based on the intermediate result it sees during the execution, The privacy spectrum can be defined based on the probability that the claim is true. On one extreme is provably exposed where an adversary can prove that $v_i = a$ (with probability of 1). On the other extreme is absolute privacy where an adversary cannot determine the exact value of $v_i$ (with probability of 0). In between, there are possible innocence where the claim is more likely to be true, and probable innocence where the claim is less likely to be true. A particularly interesting notion is beyond suspicion where a node is no more likely to have a value that satisfies the claim than any other nodes in the system. This is also known as $m$-anonymity
as we have mentioned earlier.

A closer look at the spectrum shows that it does not capture the important differences among different claims for our data privacy concerns. Consider an adversary that makes a claim \( v_i = a \) after executing a max query \( (k = 1) \) and the probability of the claim being true is \( 1/n \), i.e. there are some node(s) in the system that have the value but none is more likely than others to have it. By the privacy spectrum, the degree of data value exposure for the node is beyond suspicion. However, if \( a = v_{\text{max}} \), where \( v_{\text{max}} \) denotes the final maximum value, it should not be considered as a privacy breach at all. This is because \( v_{\text{max}} \) is public information to all the nodes after the protocol and every node has a probability \( 1/n \) holding \( v_{\text{max}} \). On the other hand, if \( a \neq v_{\text{max}} \), then it is indeed a privacy breach because other nodes would not have known anything about the value \( a \) by just knowing \( v_{\text{max}} \).

In fact, such differences among different claims are more obvious and important for data range privacy. Intuitively, a data range exposure with a very precise (small) range is much more severe than those with a large range. For example, consider the case where an adversary is able to prove \( v_i \leq a \). By the privacy spectrum, node \( i \) has provable exposure regarding its data range. However, the severity of the privacy breach actually varies (decreases as \( a \) increases). At the extreme, if \( a = v_{\text{max}} \), it should not be considered as a privacy breach at all because \( v_i \leq v_{\text{max}} \) is known to all the nodes after the final result of \( v_{\text{max}} \) is returned.

We propose a general metric - loss of privacy - to characterize how severe a data exposure is by measuring the relative loss in the degree of exposure. Let \( R \) denote the final result set after the execution and \( IR \) denote the intermediate result set during the execution. Let \( P(C|R, IR) \) denote the probability of \( C \) being true given the final result and the intermediate results, and similarly, \( P(C|R) \) the probability given only the final result. We define Loss of Privacy (LoP) in Equation 24. Intuitively, this gives us a measure of the additional information an adversary may obtain given the knowledge of the intermediate result besides the final query result.

\[
\text{LoP} = |P(C|R, IR) - P(C|R)|
\]  

(24)
We illustrate the metric for data value privacy in the context of topk queries, where \( C \) is in the form of \( v_i = a \) and \( R \) is the final topk values denoted as \( TopK \). If \( a \in TopK \), every node has the same probability to hold \( a \) so we have \( P(v_i = a|TopK) = \frac{1}{n} \). Otherwise \( (a \notin TopK) \), it is close to impossible for an adversary to guess the exact value of a node given only the final result. This is especially true when the data domain is large enough, because a node can take any of the values in the data domain. So we approximate \( P(v_i = a|TopK) \) with 0. Thus the LoP is slightly smaller when \( a \in TopK \). In the cases where all an adversary knows is that some node in the system has a value equal to \( a (a \in TopK) \), we have \( P(v_i = a|TopK,IR) = \frac{1}{n} \) and \( LoP = 0 \).

Given the definition of \( LoP \) for a single data item at a single node, we define \( LoP \) for a node as the average \( LoP \) for all the data items used by a node in participating the protocol. Intuitively, when nodes participate the protocol with their local topk values, the more values that get disclosed, the larger the \( LoP \) for the node. We measure the privacy characteristics for the system using the average \( LoP \) of all the nodes.

Other alternative metrics based on information theory [7] can be also used. We would like to explore those in our future work.

### 5.3 The Decentralized Protocol

In this section we describe a decentralized computation protocol for multiple organizations to compute topk queries over \( n \) private databases (nodes) with minimum information disclosure from each organization.

Bearing the privacy goal in mind, we identify two important principles for our protocol design. First, the output of the computation at each node should prevent an adversary from being able to determine the node’s data value or data range with any certainty. Second, the protocol should be able to produce the correct final output of a topk query (effectiveness) in a small and bounded number of rounds of communication among the \( n \) nodes (efficiency). Using these principles as the design guidelines, we propose a probabilistic protocol with a randomized local algorithm for topk queries across \( n \) private databases \( (n \geq 3) \). To facilitate the discussion of our protocol, we first present a naive protocol as the intuitive motivation.
and then describe the rational and the algorithmic details of our decentralized probabilistic protocol.

5.3.1 A Naive Protocol

Consider a group of $n$ databases who wish to select the max value ($k = 1$) of a common attribute. A straightforward way to compute the result without a central server is to have the nodes arranged in a ring in which a global value is passed from node to node along the ring. The first node sends its value to its successor. The next node computes the current max value between the value it gets from its predecessor and its own value and then passes the current max value to its successor. At the end of the round, the output will be the global max value.

Clearly, the scheme does not provide good data privacy. First, the starting node has provable exposure to its successor regarding its value. Second, the nodes that are close to the starting node in the ring have a fairly high probability disclosing their values. A randomized starting scheme can be used to protect the starting node and avoid the worst case but it would not help with the average data value disclosure of all the nodes on the ring. In addition, every node $i$ ($1 \leq i \leq n$) suffers provable exposure to its successor regarding its data range, i.e. the successor knows for sure that node $i$ has a value smaller than the value it passes on. This leads us to consider alternative protocols for better privacy preservation.

In the rest of this section, we present our probabilistic protocol. We first give a brief overview of the key components of the protocol and then use the max (min) queries (the top$k$ queries with $k = 1$) to illustrate how the two design principles are implemented in the computation logic used at each node (private database) to achieve the necessary minimum disclosure of private information (our privacy goal).

5.3.2 Protocol Structure

The protocol is designed to run over a decentralized network with a ring topology, and consists of the node communication scheme, the local computation module and initialization module at each node.

Ring Topology. Nodes are mapped into a ring randomly. Each node has a predecessor
and successor. It is important to have the random mapping to reduce the cases where two colluding adversaries are the predecessor and successor of an innocent node. We will discuss more on this in Section 5.4.

*Communication protocol.* The communication among the nodes is from a node to its successor. Encryption techniques can be used so that data are protected on the communication channel. In case there is a node failure on the ring, the ring can be reconstructed from scratch or simply by connecting the predecessor and successor of the failed node.

*Local computation module.* The local algorithm is a standalone component that each node executes independently. Nodes follow the *semi-honest* model and executes the algorithm correctly.

*Initialization module.* The initialization module is designed to select the starting node among the $n$ participating nodes and then initialize a set of parameters used in the local computation algorithms.

In this dissertation we do not handle the data schema heterogeneity issues. We assume that the database schemas and attribute names are known and are well matched across $n$ nodes. Readers who are interested in this issue may refer to [30] for some approaches to the problem of schema heterogeneity.

### 5.3.3 Privacy Preserving Max Selection

Before going into details of the protocol, we first present the local computation component of the protocol for $\max(\min)$ queries (the special case of $\top k$ with $k = 1$) over $n$ private databases. This will help readers understand the key ideas and techniques used in our protocol design. We describe the general protocol for $\top k$ queries in next subsection.

The intuitive idea of using a probabilistic protocol is to inject some randomization into the local computation at each node, such that the chance of data value disclosure at each node is minimized and at the same time the eventual result of the protocol is guaranteed to be correct. Concretely, the protocol performs multiple rounds in which a global value is passed from node to node along the ring. A randomization probability is associated with each round and decreased in the next round to ensure that the final result will be produced
in a bounded number of rounds. During each round, nodes inject certain randomization in their local computation with the given probability. The randomization probability is eventually decreased to 0 so that the protocol outputs the correct result.

**Randomization Probability.** We first define the randomization probability. It starts with an initial probability denoted as $p_0$ in the first round and decreases exponentially with a dampening factor denoted as $d$, so that it tends to 0 with sufficient number of rounds. Formally, the randomization probability for round $r$ denoted as $P_r(r)$ is defined as follows:

$$P_r(r) = p_0 * d^{r-1}$$

(25)

**Randomized Algorithm.** Each node, upon receiving the global value from its predecessor, performs the local randomized algorithm, and passes the output to its successor. The core idea of this algorithm is to determine when (the right time) to inject randomization and how much (the right amount of randomization) in order to implement the two design principles of the protocol: namely, the output of the algorithm should prevent an adversary from inferring the value or range of the data that the node holds with any certainty; and the randomized output should not generate potential errors that lead to incorrect final output of the protocol.

**Algorithm 8** Local Algorithm for Max Protocol (executed by node $i$ at round $r$)

- **INPUT:** $g_{i-1}(r), v_i$  
- **OUTPUT:** $g_i(r)$  
- $P_r(r) \leftarrow p_0 * d^{r-1}$  
- if $g_{i-1}(r) \geq v_i$ then  
  - $g_i(r) \leftarrow g_{i-1}(r)$  
- else  
  - with probability $P_r$: $g_i(r) \leftarrow$ a random value between $[g_{i-1}(r), v_i]$  
  - with probability $1 - P_r$: $g_i(r) \leftarrow v_i$

end if

A sketch of the randomized algorithm is given in Algorithm 8 for node $i$ at round $r$. The algorithm takes two inputs: (1) the global value node $i$ receives from its predecessor $i-1$ in round $r$, denoted as $g_{i-1}(r)$, and (2) its own value, denoted as $v_i$. The algorithm compares these two input values and determines the output value, denoted as $g_i(r)$, in the following two cases. First, if the global value $g_{i-1}(r)$ is greater than or equal to its own value $v_i$, node $i$ simply returns the current local maximum value ($g_{i-1}(r)$ in this case). There is no
need to inject any randomization because the node does not expose its own value in this case. Second, if $g_{i-1}(r)$ is smaller than $v_i$, instead of always returning the current local maximum value ($v_i$ in this case), node $i$ returns a random value with probability $P_r(r)$, and only returns $v_i$ with probability $1 - P_r(r)$. The random value is generated uniformly from the range $[g_{i-1}(r), v_i]$. Note that the range is open at $v_i$ to warrant that the node will not return $v_i$ when we want to return a constrained random value instead of the actual $v_i$.

Such randomization has a number of important properties. First, it successfully prevents an adversary from deducing the value or range of $v_i$ with any certainty. This is because the output of node $i$ can be either a random value, or the global value passed by the predecessor of node $i$, or its own value $v_i$. Second, the global value monotonically increases as it is passed along the ring, even in the randomization case. Recall the case when randomization is injected, the random value output $g_i(r)$ can be smaller than $v_i$ but has to be greater than or equal to $g_{i-1}(r)$, which ensures that the global value keeps increasing. This monotonic increasing property further minimizes the need for other nodes after node $i$ to have to disclose their own values because they can simply pass on the global value if it is greater than their own values. Finally, the randomized value will not generate any potential errors for the protocol because it is always smaller than $v_i$ and thus smaller than the global maximum value. It will be replaced by the value that is held either by the node $i$ itself or any other node that holds a greater value in a later round as the randomization probability decreases. We will analyze the correctness and data value privacy of the protocol formally in Section 5.4.

**Protocol Details.** We now walk through the protocol by describing the initiation process, the communication scheme, combined with the local computation logic.

At the initiation state, every node in the network sorts their values and takes the local max value to participate the global max selection. The protocol randomly chooses a node from the $n$ participating nodes, say indexed by $i$ with $i = 1$. In addition, the initialization module will set the default global value $g_0(1)$ to the lowest possible value in the corresponding data domain, and initialize the randomization probability with $p_0$, the dampening
factor $d$ (recall Section 3.2), and the round counter $r$. The key idea of using a randomized selection scheme for starting node is to preserve the anonymity of the starting node so an adversary does not know where the protocol start from and hence protecting the starting node.

Upon the completion of the initiation process, the local computation module is invoked at node $i$. Each node $i$, upon receiving the global value $g_{i-1}(r)$ from its predecessor at round $r$, executes the local computation algorithm, and passes the output $g_i(r)$ to its successor. At the end of each round $r$ ($r \geq 1$), the last node $n$ passes the current global value $g_n(r)$ to the first node, which serves as the input $g_0(r+1)$ at the first node in round $r+1$. The protocol terminates at the starting node after a sufficient number of rounds. We will discuss how to determine the number of rounds needed and what we mean by sufficient in Section 5.4. It is interesting to note that if we set the initial randomization probability to be 0 ($p_0 = 0$), the protocol is reduced to the naive deterministic protocol.

![Privacy Preserving Max Protocol: Illustration](image)

**Figure 33:** Privacy Preserving Max Protocol: Illustration

Figure 33 shows an example walk-through of the protocol over a network of 4 nodes, initialized with $p_0 = 1$ and $d = 1/2$. Assume the protocol starts from node 1 with the initial global value $g_0(1) = 0$. In the first round ($r = 1$), the randomization probability $P_r(1)$ is initialized to 1, so if a node receives a value smaller than its own value, it will always return a random value between the received value and its own value. As a result, node 1 returns a random value between $[0,30)$, say 16. Node 2 passes 16 to node 3 because it is greater than its own value 10. Node 3 returns a random value between $[16,40)$, say 25, since value 16 is smaller than its own value 40. Node passes value 25 to the first node because it is
greater than its own value 20. In the second round \( r = 2 \), the randomization probability \( P_r(2) \) decreases to \( 1/4 \) according to equation 25. As a result, node 1 returns its own value 30. Node 2 passes on value 30. Node 3 returns a random value between \([30,40]\), say 32. Node 4 passes on value 32. In the third round \( r = 3 \), the randomization probability \( P_r(3) \) decreases to \( 1/4 \). Node 1 and Node 2 both pass on the value 32. Node 3 finally returns its own value 40 and node 4 passes on the value 40. In the termination round all nodes simply passes on the final result.

This example illustrates how our probabilistic protocol works and why our protocol ensures that each node retains good privacy about the exact value and the range of their data. It is important to note that the random selection scheme for the starting node plays an important role for preserving good privacy of the starting node. For instance, in the above example, if node 1 was known as the starting node, then upon receiving 16 from node 1 in the first round, node 2 knows for sure that node 1 has a value greater than 16, leading to the data range exposure for node 1.

We provide an analytical model to formally study the correctness and data value privacy of the protocol in Section 5.4 and report the result of our experimental evaluation in Section 5.5.

### 5.3.4 Privacy Preserving Top-k Selection

Now we describe the general protocol for topk selection. It works similarly as the max selection protocol \( k = 1 \) in terms of the probabilistic scheme. The complexity of extending the protocol from max to general topk lies in the design of the randomized algorithm.

At the protocol initial step, each node first sorts its values and takes the local set of topk values as its local topk vector to participate in the protocol, since it will have at most \( k \) values that contribute to the final topk result. Similar to the max selection protocol, the initialization module randomly picks a node from the \( n \) participating nodes as the starting node, initializes the global topk vector to the lowest possible values in the corresponding data domain, sets the round counter \( r \), and initializes the randomization probability \( p_0 \) and the dampening factor \( d \) (recall Equation 25 in Section 5.3.3).
The protocol performs multiple rounds in which a current global topk vector is passed from node to node along the ring network. Each node $i$, upon receiving the global vector from its predecessor at round $r$, performs a randomized algorithm and passes its output to its successor node. The starting node terminates the protocol after a sufficient number of rounds.

**Randomized Algorithm.** The randomized algorithm is the key component of the probabilistic topk selection protocol. We want the algorithm to have the same properties as those of the max selection algorithm (Algorithm 8) when deciding the right time and the right amount of randomization to inject, namely, to guarantee the correctness on one hand and minimize the data value disclosure on the other hand. For example, we can use the same idea of generating random values and inject them into the output of the global topk vector at node $i$ ($1 \leq i \leq n$) in order to hide the node’s own values. However, with $k$ values in the local topk vector, we need to make sure that the randomly generated values will eventually be shifted out from the final global topk vector. In other words, it is not as straightforward as in the max selection algorithm where a random value less than a node’s value will be replaced eventually.

**Algorithm 9** Local Algorithm for Topk Protocol (executed by node $i$ at round $r$)

```
INPUT: $G_{i-1}(r), V_i$  OUTPUT: $G_i(r)$
$P_r(r) \leftarrow p_0 \ast d^{r-1}$
$G'_i(r) = \text{topK}(G_{i-1}(r) \cup V_i)$
$V'_i \leftarrow G'_i(r) - G_{i-1}(r)$
$m \leftarrow |V'_i|$
if $m = 0$ then
$G_i(r) \leftarrow G_{i-1}(r)$
else
with probability $1 - P_r(r)$: $G_i(r) \leftarrow G'_i(r)$
with probability $P_r(r)$:
$G_i(r)[1 : k - m] \leftarrow G_{i-1}(r)[1 : k - m]$
$G_i(r)[k-m+1 : k] \leftarrow \text{sorted list of } m \text{ random values from } \left[ \min(G'_i(r)[k] - \delta_i, G_{i-1}(r)[k-m+1]) \right]$
end if
```

Algorithm 9 gives a sketch of a randomized algorithm for general topk selection with respect to node $i$ executing at round $r$. The input of the algorithm is (1) the global vector node $i$ receives from its predecessor $i - 1$ in round $r$, denoted as $G_{i-1}(r)$, and (2) its local topk
vector, denoted as \( V_i \). The output of the algorithm is the global vector denoted as \( G_i(r) \). Note that the global vector is an ordered multiset that may include duplicate values.

The algorithm first computes the real current top \( k \) vector, denoted as \( G'_i(r) \), over the union of the set of values in \( G_{i-1}(r) \) and \( V_i \), say, using a merge sort algorithm. It then computes a sub-vector of \( V_i \), denoted as \( V'_i \), which contains only the values of \( V_i \) that contribute to the current top \( k \) vector \( G'_i(r) \) by taking a set difference of the set of values in \( G'_i(r) \) and \( G_{i-1}(r) \). Note that the union and set difference here are all multiset operations. The algorithm then works under two cases.

Case 1: The number of elements in \( V'_i \), \( m \), is 0, i.e., node \( i \) does not have any values to contribute to the current top \( k \). In this case, node \( i \) simply passes on the global top \( k \) vector \( G_{i-1}(r) \) as its output. There is no randomization needed because the node does not expose its own values.

![Diagram](image_url)

**Figure 34:** Local Algorithm of Privacy Preserving Top \( k \) Protocol: Illustration

Case 2: Node \( i \) contributes \( m (0 < m \leq k) \) values in the current top \( k \). Figure 34 gives an illustrative example where \( m = 3 \) and \( k = 6 \). In this case, node \( i \) only returns the real current top \( k \) (\( G'_i(r) \)) with probability \( 1 - P_i(r) \). Note that a node only does this once, i.e., if it inserts its values in a certain round, it will simply pass on the global vector in the rest of the rounds. With probability \( P_i(r) \), it copies the first \( k - m \) values from \( G_{i-1}(r) \) and generate last \( m \) values randomly and independently from \([\min(G'_i(r)[k] - \delta, G_{i-1}(r)[k-m+1]), G'_i(r)[k]]\), where \( G'_i(r)[k] \) denotes the \( k \)th (last) item in \( G'_i(r) \), \( G_{i-1}(r)[k-m+1] \) denotes the \( k - m + 1 \)th item in \( G_{i-1}(r) \), and \( \delta \) denotes a minimum range for generating the random
values. The reason for generating $m$ random values is because only the last $m$ values in the output are guaranteed to be shifted out in a later round when the node inserts its real values if the global vector has not been changed by other nodes. The range is designed is such a way that it increases the values in the global vector as much as possible while guaranteeing the random values do not exceed the smallest value in the current top $k$ so they will be eventually replaced. In an extreme case when $m = k$, the current top $k$ vector is equal to $V_i$, it will replace all $k$ values in the global vector with $k$ random values, each randomly picked from the range between the first item of $G_{i-1}(r)$ and the $k$th (last) item of $V_i$.

It is worth noting that when $k = 1$ the local top $k$ selection algorithm becomes the same as the local algorithm for max protocol. We report our experimental evaluation on the correctness and privacy characteristics of the general protocol in Section 5.5.

5.4 Analysis

We conducted a formal analysis on the max protocol in terms of its correctness, efficiency, and privacy characteristics.

5.4.1 Correctness

Let $g(r)$ denote the global value at the end of round $r$ and $P(g(r) = v_{\text{max}})$ denote the probability that $g(r)$ is equal to the global max value $v_{\text{max}}$. At round $j (1 \leq j \leq r)$, if the global value has not reached $v_{\text{max}}$, the nodes who own $v_{\text{max}}$ have a probability $1 - P_r(j)$ to replace the global value with $v_{\text{max}}$. Once the global value reaches $v_{\text{max}}$, all nodes simply pass it on. So after round $r$, the global value will be equal to $v_{\text{max}}$ as long as one of the nodes that owns $v_{\text{max}}$ has replaced the global value with $v_{\text{max}}$ in any of the previous rounds. Thus the probability of the protocol returning the global maximum value after round $r$ can be computed as $P(g(r) = v_{\text{max}}) \geq 1 - \prod_{j=1}^{r} P_r(j)$. If we substitute $P_r(j)$ with $p_0 * d^{j-1}$ by Equation 25, we can derive the following equation:

$$P(g(r) = v_{\text{max}}) \geq 1 - p_0^r * d^{\frac{r(r-1)}{2}}$$

(26)
Equation 26 shows that, for any $0 < p_0 \leq 1$ and $0 < d < 1$, the precision bound increases monotonically with increasing $r$. For any given number of nodes, we can make the computed global value equal to the global max value with a probability very close to 1 by increasing the number of rounds.

Figure 35: Precision Guarantee with Number of Rounds

Figure 35(a) and (b) plot the precision bound in equation 26 with increasing number of rounds ($r$) for varying initial randomization probability ($p_0$) and dampening factor ($d$) respectively. We can see that the precision increases with the number of rounds. A smaller $p_0$ with a fixed $d$ results in a higher precision in the earlier round and reaches the near-perfect precision of 100% faster. A smaller $d$ with a fixed $P_0$ makes the protocol reach the near-perfect precision of 100% even faster.

5.4.2 Efficiency

Now we analyze the efficiency of the protocol in terms of the computation and communication cost. The computation at each node in the protocol does not involve any cryptographic operations and should be negligible compared to the communication cost over a network of $n$ nodes. The communication cost is determined by two factors. The first is the cost for a single round which is proportional to the number of nodes on the ring. The second is the number of rounds that is required for a desired precision. For any $\epsilon(0 < \epsilon < 1)$, we can determine a minimum number of rounds, denoted by $r_{min}$, such that the result is equal to the global max value with the probability greater than or equal to $1 - \epsilon$. Since we have $P(g(r) = v_{max}) \geq 1 - p_0 * d^{-\frac{r-1}{2}} \geq 1 - p_0 * d^{-\frac{r-1}{2}}$ from Equation 26, we can ensure $P(g(r) = v_{max}) \geq 1 - \epsilon$ by requiring $1 - p_0 * d^{-\frac{r-1}{2}} \geq 1 - \epsilon$. We solve this equation and
derive a minimum number of rounds for the desired precision \((1 - \epsilon)\) as follows:

\[
r_{\min} = \left\lceil \frac{1}{2} \left( 1 + \sqrt{8 \cdot \frac{\log(\epsilon/p_0)}{\log d}} - 1 \right) \right\rceil
\]  

(27)

**Figure 36:** Required Number of Rounds with Precision Guarantee

We can see that the minimum number of rounds \(r_{\min}\) scales well with the desired precision \((1 - \epsilon)\) in the order of \(O(\sqrt{\log(e)}\). Figure 36(a) and (b) plot the minimum number of rounds in Equation 27 for varying error bound \(\epsilon\) with varying initial randomization probability \(p_0\) and dampening factor \(d\) respectively. Note that the X axis is of logarithmic scale. We can see that the protocol scales well with increasing desired precision (decreasing \(\epsilon\)). In addition, a smaller \(p_0\) and a smaller \(d\) are desired for better efficiency with \(d\) having a larger effect on the reduction of the required number of rounds.

It is important to note that the minimum number of rounds is independent of the number of nodes. Hence, the overall communication cost is proportional to the number of nodes. One possible way to improve the efficiency for a system with a larger number of nodes is to break the set of \(n\) nodes into a number of small groups and have each group compute their group maximum value in parallel and then compute the global maximum value at designated nodes, which could be randomly selected from each small group.

### 5.4.3 Data Value Privacy

In addition to ensuring correct output and increasing efficiency of the protocol, another important goal of the protocol is preserving the data privacy of individual participating nodes in the network. The communication between nodes consists of sending the current global value from one node to its successor. An adversary may utilize the value it receives
from its predecessor to try to gain information about the data its predecessor and other nodes hold. We dedicate this section to analyzing the loss of data value privacy in the protocol using the metric we proposed in Section 5.2.

Without loss of generality, we assume the protocol starts from node 1. We now analyze the loss of privacy for node $i$ with value $v_i$. Since node $i$ passes its current global value $g_i(r)$ to its successor $i + 1$ in round $r$, all the successor can do with respect to the exact data value node $i$ holds is to guess that $v_i = g_i(r)$. Recall Equation 24 in Section 5.2, the loss of privacy is defined as the relative probability of a node holding a particular value with and without the intermediate result. Let $P(v_i = g_i(r)|g_i(r), v_{max})$ denote the probability that node $i$ holds the value $g_i(r)$ with the knowledge of the intermediate result $g_i(r)$ and $P(v_i = g_i(r)|v_{max})$ denote the probability without it. If $g_i(r) = v_{max}$, we have $P(v_i = g_i(r)|v_{max}) = 1/n$ as all nodes have the same probability holding the global maximum value. Otherwise, we approximate $P(v_i = g_i(r)|v_{max})$ with 0 as we have discussed earlier in Section 5.2. Now let us look at $P(v_i = g_i(r)|g_i(r), v_{max})$ for both naive protocol and probabilistic protocol and derive the Loss of Privacy (LoP).

**Naive Protocol.** In the naive protocol where only one round is needed, the global value $g_i$ that node $i$ passes on is the current maximum value of all its previous nodes and itself. Since all of them have the same probability to hold the current maximum value, so we have $P(v_i = g_i(r)|g_i(r), v_{max}) = 1/i$. Thus the data value loss of privacy for node $i$ in naive protocol is $1/iC1/n$ if $g_i = v_{max}$ and $1/i$ otherwise.

It is clear that the loss of privacy for node $i$ in the naive protocol depends on its position. Nodes closer to the starting node suffer a larger loss of privacy. In the worst case, the starting node ($i = 1$) has provable exposure of its value to its successor. By average, the loss of privacy is greater than $\sum_{i=1}^{n}(1/i - 1/n)/n$. Using the inequality bound for $\sum_{i=1}^{n} 1/i$ (the nth Harmonic number [62]), we can derive the average LoP bound for the naive protocol in Equation 28. We can see that it is fairly high especially when $n$ is small.

$$\text{LoP}_{\text{Naive}} > \frac{\ln n}{n}$$  \hspace{1cm} (28)

**Probabilistic Protocol.** The probabilistic max selection protocol requires multiple rounds.
Since aggregating the values a node passes on in different rounds does not help with determining its exact data value, though it may help with determining the probability distribution of the value, we first compute the loss of privacy for node $i$ at each round $r$ and then take the highest result in all the rounds as final loss of privacy for node $i$.

Recall the probabilistic computation in Algorithm 8, a node only replaces the global value with its own value with probability $1 - P_r(r)$ when the value it receives is smaller than its own value. Thus we have $P(v_i = g_i(r)|v_{max}) = P(v_i > g_{i-1}(r)) * (1 - P_r(r)) + P(v_i = g_{i-1}(r))$. We performed an analysis on the expected value for $g_i(r)$ and derived an approximate lower bound of expected LoP for node $i$ in round $r$. By taking the highest (maximum) LoP of all rounds for each individual node, we derive the average expected LoP for all the nodes in Equation 29.

$$E(LoP_{probabilistic}) \leq max_r \left( \frac{1}{2r-1} * (1 - p_0 * d^{r-1}) \right) \quad (29)$$

![Expected Loss of Privacy in Different Rounds](image)

**Figure 37:** Expected Loss of Privacy in Different Rounds

From Equation 29, we can see that the expected loss of privacy for the probabilistic protocol depends on how we choose the randomization parameters. Also intuitively, the highest (peak) loss of privacy may happen at different rounds with different randomization parameters. Figure 37 plots the behavior of the bound inside the $max$ function with varying randomization parameters. Figure 37(a) shows the effect of $p_0$ with $d$ set to $1/2$. It is interesting to see that $p_0$ plays an important role in the loss of privacy. A larger $p_0$ results in a lower loss of privacy in the first round. The reason is quite intuitive since a larger $p_0$ implies that more nodes have injected randomized values instead of returning the real
current max value in the computation. With a smaller $p_0$, the loss of privacy gradually decreases from the peak of loss as the protocol converges. With a larger $p_0$, such as $p_0 = 1$ as shown, the loss of privacy starts with 0 in the first round and increases in the second round to the peak of loss, and then gradually decreases. If we compare the peak loss of privacy in different rounds, we conclude that a larger $p_0$ provides a better privacy. Figure 37(b) shows the effect of $d$ with $p_0$ set to 1. We see that a larger $d$ corresponds to a lower loss of privacy, starting from the second round, though with a small margin. Overall, by tuning the parameters $p_0$ and $d$, we can keep the loss of privacy very low. Our experimental evaluation in Section 5.5 confirms with our analytical results regarding the loss of privacy.

Now we briefly discuss the loss of privacy under the scenario where the predecessor and the successor of node $i$ happen to collude with each other. Assuming that an adversary has intermediate results of both $g_{i-1}(r)$ and $g_i(r)$, we have $P(v_i = g_i(r)|g_{i-1}(r), g_i(r), v_{max}) = 1 - P_t(r)$ when $g_{i-1}(r) < g_i(r)$. Knowing this still does not give the adversary any certainty in determining the data value of node $i$, especially in the beginning rounds when $P_t(r)$ is large. Intuitively, when $P_t(r)$ gets close to 0, $g_{i-1}(r)$ should be already getting close to $v_{max}$ so there is very small chance for the data at node $i$ to be disclosed. It is interesting to note though, if node $i$ happens to hold $v_{max}$ then it will be susceptible to provable exposure if it has two colluding neighbors. One technique to minimize the effect of collusion is for a node to ensure that at least one of its neighbors is trustworthy. This can be achieved in practice by having nodes arrange themselves along the network ring(s) according to certain trust relationships such as digital certificate based trust [16] combined with reputation based trust that we have explored in this dissertation. Further, we can extend the probabilistic protocol by performing the random ring mapping at each round so that each node will have different neighbors at each round.

5.5 Experimental Evaluation

This section presents a set of initial results from our experimental evaluation of the protocols in terms of correctness and privacy characteristics.
5.5.1 Experiment Setup

Table 4: Primitive Protocol: Experiment Parameters

<table>
<thead>
<tr>
<th>Param.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td># of nodes in the system</td>
</tr>
<tr>
<td>$k$</td>
<td>parameter in topk</td>
</tr>
<tr>
<td>$p_0$</td>
<td>initial randomization prob.</td>
</tr>
<tr>
<td>$d$</td>
<td>dampening factor for randomization prob.</td>
</tr>
</tbody>
</table>

The system consists of $n$ nodes. The attribute values at each node are randomly generated over the integer domain $[1, 10000]$. We experimented with various distributions of data, such as uniform distribution, normal distribution, and zipf distribution. The results are similar so we only report the results for the uniform distribution. The experiment proceeds by having the nodes compute top$k$ values using the probabilistic protocol. We evaluate the accuracy and privacy properties. Each plot is averaged over 100 experiments. Table 4 lists the main parameters for the experiments.

5.5.2 Precision of Max Selection

![Figure 38: Precision of Max Selection with Number of Rounds](image)

We first verify the correctness of the probabilistic max protocol ($k = 1$). Figure 38(a) and (b) show the precision with increasing number of rounds ($r$) for different initial randomization probability ($p_0$) and dampening factor ($d$) respectively. We see that the experimental results match the analytical bounds in Figure 35. The precision reaches to 100% as the number of rounds increases. A smaller $p_0$ results in a higher precision in the first round and makes
the precision go up to 100% faster as the number of rounds increases, though with a small margin. A smaller $d$ reaches the perfect precision of 100% much faster.

### 5.5.3 Privacy Characteristics of Max Selection

We evaluate the privacy characteristics of the protocol in terms of their data value loss of privacy. In particular, we want to answer a number of questions. What is the loss of privacy during the execution of the algorithm? How does the number of nodes affect the privacy characteristics? How do the randomization parameters affect the privacy characteristics and how to select them? How does the protocol compare to the naive protocol?

**Loss of Privacy in Different Rounds.** We first study the loss of privacy of the protocol in each round during the execution with different randomization parameters. We experimented with different number of nodes and the trends in different rounds are similar but most pronounced with a small number of nodes. So we only report the results for $n = 4$ to show the different loss of privacy in different rounds and will present another set of experiments later to show the effect of varying number of nodes.

![Figure 39: Loss of Privacy for Max Selection in Different Rounds](image)

Figure 39(a) and (b) show the average data value loss of privacy for all nodes in different rounds with varying initial randomization probability ($p_0$) and dampening factor ($d$) respectively. The result matches our analysis in Section 5.4. With a smaller $p_0$, the highest loss of privacy happens in the first round and it gradually decreases as the protocol converges. With a larger $p_0$ (e.g., $p_0 = 1$), the loss of privacy is zero in the first round and reaches the peak in the second round and then gradually decreases. If we look at the peak loss of privacy, a larger $p_0$ provides a better privacy. In Figure 39(b), all three cases ($d = 1,$
$d = 1/2$, $d = 1/4$) start with zero loss of privacy in the first round, and increase to the highest (peak loss) in the second round, and decreases as the protocol converges. A smaller $d$ results in a higher peak loss of privacy.

In this set of experiments, we have shown the loss of privacy in different rounds during the execution. For the rest of the experiments we will take the highest (peak) loss of privacy among all the rounds for a given node to measure its overall loss of privacy, because that gives us a measure of the highest level of knowledge an adversary can obtain regarding the node’s data value.

**Effect of Number of Nodes.** We now report the experiments showing how the number of nodes affects the loss of privacy of the protocol.

![Figure 40: Loss of Privacy for Max Selection with Different Number of Nodes](image)

Figure 40(a) and (b) show the average data value loss of privacy for all nodes with varying initial randomization probability ($p_0$) and dampening factor ($d$) respectively. We can see that the loss of privacy decreases with increasing number of nodes. This is very intuitive because the larger the number of nodes, the faster the global value increases and thus the less probability the nodes have to disclose their own values. Again, the result shows that a smaller $p_0$ and $d$ provide a better privacy.

**Selection of Randomization Parameters.** This set of experiments is dedicated to study the effect of randomization parameters on both privacy characteristics and efficiency of the protocol. Recall the experiments described so far and our analysis in Section 5.4, a smaller $p_0$ and $d$ provide better privacy but requires more cost in terms of number of rounds required. Our design goal is to increase the efficiency while minimizing the loss of
privacy. Put differently, we want to select a pair of $p_0$ and $d$ parameters that gives us the best tradeoff between privacy and efficiency.

![Graph showing the tradeoff between privacy and efficiency with varying randomization parameters.](image)

**Figure 41:** Tradeoff between Privacy and Efficiency with Varying Randomization Parameters

Figure 41 shows the loss of privacy on X axis and the cost in terms of number of rounds for a given precision guarantee ($\epsilon = 0.001$) on Y axis for varying randomization parameter pairs $(p_0, d)$. We can see that $p_0$ has a dominating effect on the loss of privacy while $d$ has a dominating effect on efficiency. The $(p_0, d)$ pair of $(1, 1/2)$ in the lower left corner gives a nice tradeoff of privacy and efficiency. Therefore, we will use $p_0 = 1$ and $d = 1/2$ as our default parameters for the rest of the experiments.

**Comparison of Different Protocols.** We have discussed the naive protocol with fixed starting node, and our probabilistic protocol. The experiments reported below compare the probabilistic protocol with the naive protocol. For comparison purposes, we include the anonymous naive protocol which uses a randomized starting scheme, instead of fixed starting node, to provide the anonymity of the starting node. We show both average and worst case loss of privacy for different nodes in the system. By worst case, we mean highest loss of privacy among all the nodes and it typically happens at the starting node in the fixed starting scheme. Our goal is to show the effectiveness of our probabilistic protocol over the naive ones and the benefit of randomly selecting the starting node.

Figure 42(a) and (b) show the average and worst case data value loss of privacy for all nodes with different number of nodes $(n)$ respectively. We make a few interesting observations. First, the anonymous starting scheme has the same average LoP as the naive protocol but avoids the worst case scenario. This can be seen in Figure 42(b) where the naive
protocol suffers a loss of privacy close to 100% (at the starting node) while the anonymous protocol does not change significantly from the average case in Figure 42(a). Second, the probabilistic protocol achieves significantly better privacy than the naive protocols. It is close to 0 in most cases. Finally, all of the protocols have a decreasing loss of privacy as the number of nodes increases. Interestingly, when the number of nodes is sufficiently large, the anonymous naive protocol performs reasonably well compared to the probabilistic protocol. However, most of the privacy preserving data integration will be among tens or hundreds of nodes with membership controls. A network of size on the order of thousands seldom happens.

5.5.4 Precision of Top-k Selection

We have presented results for max protocol so far. Now we evaluate the general topk protocol in terms of its correctness and privacy characteristics. In addition to running the same set of experiments for max protocol, we also run a set of experiments with varying k. Since most of the results we obtained are similar to those for max protocol, we only report in these two subsections the results for general topk protocol with varying k to show the effect of k.

We first verify the correctness of topk protocol. In order to evaluate the precision of top-

k selection, we first define the precision metric we use. Assume TopK is the real set of top-k values and R is the set of top-k values returned. We define the precision as \(|R \cap TopK|/K\). Figure 43 shows the precision of the topk protocol with increasing number of rounds (r) for varying k. The precision reaches to 100% in all lines after a sufficient number of rounds.
The effect of $k$ on the convergence is not significant. We also ran experiments for varying $n$ with $k > 1$ and the result did not show any significant effect.

5.5.5 Privacy Characteristics of Top-$k$ Selection

Now we report the loss of privacy for the general top-k protocol with varying $k$ and its comparison to the naive protocol.

Figure 44: Comparison of Loss of Privacy with $k$

Figure 44(a) and (b) show the average and worst case data value loss of privacy for all nodes with varying $k$. We can make a few interesting observations.

We see that the probabilistic protocol achieves significantly better privacy than the naive protocols. Interestingly, the probabilistic protocol has an increasing loss of privacy as $k$ increases. An intuitive explanation is that the larger the $k$, the more information a node exposes to its successor and hence the larger the loss of privacy.
5.6Summary

We have presented a set of probabilistic protocol for max, min, and top-k selections across multiple private databases as a building block for performing feedback aggregation and data integration in general across private databases. We formalized the notion of loss of privacy in terms of information revealed and developed an efficient decentralized probabilistic protocol, which aims at selecting top-k data items across multiple private databases with minimal information disclosure. We evaluated the correctness and privacy characteristics of the proposed protocol through both formal analysis and experimental evaluations.
CHAPTER VI

PRESERVING FEEDBACK PRIVACY: AGGREGATION

We have described a set of protocols for primitive operations such as max, min, topk in Chapter 5 as a building block for making the trust framework privacy conscious. There are also other primitive protocols that are proposed recently such as a secure summation protocol [21] and kth element [6] protocol. In this chapter, we show how we can build a privacy preserving trust framework through privacy preserving feedback aggregation on top of the primitive privacy preserving protocols. In particular, as an example, we model a set of reputation algorithms we considered earlier as kNN classification problem and present a privacy preserving kNN classification framework. The framework is also motivated by and applicable to the general kNN classification problem. Thus the rest of the chapter is presented with a general context and the particular relevance to the feedback aggregation in reputation trust framework is discussed in places where appropriate.

6.1 Problem Statement

We are entering a highly connected and data intensive world. The information age has enabled many organizations to collect large amounts of data. Many organizations wish to discover and study interesting patterns and trends of both their own private databases and information bases of their competitors, such as learning the sales trend and sales patterns of Intel computers in US markets or Asian markets. Therefore, privacy-preserving data mining [9, 89] becomes an important enabling technology for mining data from multiple private databases provided by different and possibly competing organizations. As a field, data mining has introduced new algorithms like association rule learning, and new classification algorithms like Naive Bayes, Decision Tree and k-Nearest Neighbor classifiers, to identify interesting trends in data without assuming any a priori hypothesis on the data.

In privacy preserving data mining across multiple private databases, two or more nodes
owning confidential databases wish to run a data mining algorithm on the union of their data without revealing any unnecessary information. Privacy preserving data mining enables organizations to share statistical information of their private data while simultaneously protecting privileged information residing in individual private databases.

To see why privacy requirements play an important role in distributed data mining, consider the following scenarios [20]. Many insurance companies collect data on disease incidents, seriousness of the disease and patient background. The Center for Disease Control would like to identify disease outbreaks. One way to do this is to mine the data held by the various insurance companies for patterns that are indicative of disease outbreaks. However, commercial and legal reasons prevent the insurance companies from revealing their data. In this situation, it will be important and beneficial to have a distributed data mining algorithm that is capable of identifying potential outbreaks while respecting the privacy requirements of its participants.

Privacy preserving data mining algorithms also play an important role in industry collaborations. Industry trade groups want to identify best practices to help members. However, some practices may be trade secrets. We would like to be able to discover patterns (like “manufacturing using chemical supplies from supplier X have high failure rates”) while preserving secrets of individual organizations (like “manufacturing process Y gives low failure rates”).

The above real world problems can be modeled as data classification problems, and solved using a $k$-Nearest Neighbor classifier. A $k$-nearest neighbor classifier is a conceptually straightforward way of approximating any real valued or discrete valued classification function. In its training phase, a $k$-nearest neighbor classifier simply stores all the training examples. Generalizing beyond these examples is postponed until a new query instance must be classified. In a distributed setting, this means that the overhead of training a classifier is avoided until a query instance must be classified. Every time a new query instance is encountered, its relationship to the previously stored instances is examined and a classification is assigned to the new instance. A $k$NN classifier has many advantages over other machine learning classifiers. Instead of estimating the classification function once for the
entire instance space, the $k$NN classifier can estimate it locally and differently for each new instance to be classified. This is extremely useful in situations in which the databases of nodes are dynamic and the classification function that we wish to determine is very complex and thus cannot be represented by a single, global function, but instead can be represented by a collection of less complex local approximations.

For instance, to determine whether a pattern of disease is indicative of an outbreak, we can train a $k$NN classifier to recognize disease outbreak patterns and use it to classify a query pattern as an outbreak or not. For the second problem, the various organizations can use a $k$NN algorithm to determine the answer to their queries (such as “Is the failure rate higher than 0.5 when manufacturing using supplies from supplier X”) without revealing private information (such as “My failure rate for manufacturing using supplies from supplier X is 0.36”). However, current distributed $k$NN classifiers are not designed to guarantee the privacy requirements of the participating nodes.

The similarity based reputation computation in our reputation based trust framework can be also modeled as a classification problem. The training data is the existing feedback. Given a target peer, the users who have ratings about the target peer are previously stored instances with classification assigned based on whether their rating indicates the target peer is trustworthy. Given a source peer who is trying to evaluate the trustworthiness of the target peer, it is treated as a new query instance and the $k$NN classification algorithm can determine the $k$ nearest neighbors of the source peer based on a distance metric (the similarity metrics we have considered) and compute a personalized reputation of the target peer for the source peer. The challenge is how to perform the classification in a privacy preserving way so that peers do not disclose their own feedbacks.

One solution to privacy preserving data classification across multiple private databases using a $k$NN classifier is to have a trusted third party (TTP). The nodes send their data along with the query to the TTP, which constructs the $k$NN classifier using the data, classifies the query and sends the results back to all the nodes. However, in practice it is difficult to find a TTP which is trusted by all the nodes. Also, this solution presents a single point of failure which is the TTP. If the TTP is compromised, the privacy of all the nodes...
is compromised. To overcome the above limitations, we propose a distributed solution to the privacy preserving classification problem.

Another possibility one might consider is to construct a privacy preserving kNN classifier using secure multiparty computation techniques [35], a subject that has received significant attention in cryptography research. However, these techniques have a high communication overhead and are feasible only if the inputs are very small. In a real world scenario, the inputs to the kNN classifier would consist of massive data sets. Thus, generic secure multiparty computation protocols are too inefficient to be applied to these problems. This deficiency of secure multiparty schemes has led to the search for efficient, privacy preserving data mining algorithms. A practical idea is to look for algorithms that can provide a desired level of tradeoff between the accuracy of the classifier constructed and the stringency of the privacy requirements while maintaining efficiency.

With these design objectives in mind, we present a privacy-preserving framework for constructing a kNN classifier across multiple private databases. This framework consists of a general model for privacy preserving kNN classification, a suite of concrete algorithms for realizing this model, and a set of requirements that all privacy preserving classifiers should strive to achieve. We discuss how well our algorithm achieves the specified requirements through analytical study and experimental evaluation. To the best of our knowledge, this is the first work to show how kNN classification can be achieved in a privacy preserving manner without a centralized trusted third party. Our approach has an important trait — it offers a trade-off between accuracy, efficiency and privacy, allowing our privacy-preserving kNN classifier to be applied in a variety of problem settings and meeting different optimization criteria.

The rest of this chapter is organized as follows. In section 6.2, we present the general issues and design goals in privacy preserving data classification algorithms. In Section 6.3, we present a model for the construction of a privacy preserving kNN data mining algorithm. In Section 6.4, we present algorithms for realizing the privacy preserving kNN classifier construction according to our model. In section 6.5, we analyze our solution and discuss how well we are able to address the issues mentioned in section 6.2. In Section
6.6, we present experimental results of the evaluation of our algorithm. We summarize the chapter in Section 6.7.

6.2 Design Issues

In this section, we identify the design goals that any privacy preserving classification algorithm must seek to fulfill. These design goals also define dimensions along which any such algorithm can be evaluated.

6.2.1 Problem Definition and Threat Model

Consider a network of \( n \) nodes \((n \geq 3)\), each having a private database. Assume that we want to train a machine learning classifier on the union of these \( n \) databases, under the condition that each node wishes to reveal as little information about its database as possible during the (distributed) training of the classifier and during the possibly distributed classification of the test instances.

We note that in any multi-party computation, a malicious adversary can always alter its input. In the data classification setting, this can be very damaging since the adversary can define its input to be the empty database and thereby gain knowledge of the classifier for free, or can alter its input to invalidate the results of the classifier. In this chapter we assume that the nodes participating in our decentralized protocol for computing the \( k \)NN classification among \( n \) private databases are cooperating to achieve the common goal of data classification. Thus all nodes behave in a semi-honest manner in the sense that all nodes correctly follow the protocol specification, yet attempt to learn additional information about other nodes by analyzing the transcript of messages received during the execution of the protocol. One of our ongoing efforts is to develop a decentralized \( k \)NN classification protocol that is resilient against malicious nodes.

6.2.2 Evaluation Metrics

One of the most important dimensions along which any classification algorithm is evaluated is its accuracy. The accuracy of a classification algorithm measures its ability to correctly classify instances that it has not seen before, i.e., not present in the test set. When we
are designing a privacy preserving classification algorithm, we are interested in the relative accuracy of this algorithm as compared to a non-privacy preserving classification algorithm. Ideally, we would like the accuracy of a privacy-preserving classification algorithm to be at least as high as that of its non-privacy preserving counterpart. We note that the accuracy of a privacy preserving classifier could be lower if it utilizes randomization to ensure privacy of its computations. In this case, the design challenge is to determine the "right" amount of randomization that the algorithm should insert into its computation such that both the accuracy and privacy requirements of the users can be met.

Another important consideration in designing privacy preserving algorithms is their efficiency. Classification algorithms typically operate on databases containing very large amounts of data; thus the many secure multi-party computation protocols proposed earlier, such as [35, 36], are simply too inefficient to be used in these classification algorithms. This is especially the case when the number of nodes involved in the construction of the classifier is large. Given the above scenario, we would like the privacy preserving classification algorithm to allow for a trade-off between the accuracy of the results and the efficiency of the algorithm used.

Finally, any privacy preserving algorithm must meet the privacy requirements of its participating nodes. Ideally, during the training phase of a classification algorithm, no information other than the trained classifier should be revealed to any node. During the classification phase, no information other than the classification of the query instance should be revealed to the nodes. However, achieving the above objectives completely might make the classification algorithm very inefficient. In practice, we would like to design highly efficient classification algorithms while maintaining the minimum information disclosure.

We distinguish between two types of privacy guarantees that have been proposed in the literature: data anonymity and data privacy. The data anonymity requirement refers to the requirement that nodes may learn the fact that some actual data point is present in the database of the other nodes, but no node should be able to infer which other node this data point belongs to with certainty. The term "data privacy" has been used to refer to a stronger requirement that no node may learn about the value of any other node's data
points during the algorithm. We concentrate on achieving data privacy in this chapter.

Thinking of the design space in terms of these three dimensions presents many advantages to the designer of a privacy preserving classifier. At one end of the spectrum, we have the secure multi-party computation protocols, using which we can construct classifiers which are provably secure in the sense that they reveal the least amount of information and have the highest accuracy; however these protocols are very inefficient. At the other end of the spectrum, we have the non-privacy preserving classifier algorithms, which are highly efficient but are not secure. Thus, the challenge is to design an efficient privacy preserving classifier while sacrificing as little as possible on accuracy and privacy.

6.3 Privacy Preserving \( k \)NN Classification Model

In this section, we describe the \( k \)NN classification problem and discuss how we can solve it in a distributed, privacy preserving manner. We describe the privacy goals of our protocol and show how our classifier model achieves these privacy goals.

**Algorithm 10** Distance-weighted \( k \)NN Classification Algorithm

Training Algorithm:

- For each training example \( < x, f(x) > \), add the example to the list training - examples.

Classification Algorithm:

- Given a query instance \( x_q \) to be classified,

1. Let \( x_1, x_2, \ldots, x_k \) denote the \( k \) instances from training - examples that are nearest to \( x_q \).

2. RETURN

\[
\text{Classification}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^{k} w(d(x_q, x_i)) \cdot \delta(v, f(x_i)),
\]

where \( \delta(a, b) = 1 \) if \( a = b \), and 0 otherwise, and \( w \) is a normalized weight function of the distance \( d(x_q, x_i) \) between the points \( x_q \) and \( x_i \).

6.3.1 Classification Problem and the \( k \)NN Classifier

A classification problem consists of a set of instances, partitioned into a training set and a test set. Each instance is represented by a vector of attributes, and belongs to a class. A
classifier is a function of the attributes that returns a class value. A classifier construction algorithm takes the training set as input, and returns a classifier. The accuracy of a classifier is measured using the test set: The predictions made by the classifier for the instances in the test set are checked against the class of these instances, and the percentage of correct predictions is the accuracy of the classifier.

The k-Nearest Neighbor (kNN) classifier is one of the most basic instance-based classification methods. In an instance based classifier, when a query instance is encountered, similar instances are retrieved from memory and used to classify the query instance. In a kNN classifier, given a query instance, the $k$ nearest (according to a distance function) instances to this query instance are retrieved, and the class assigned to the query instance is the most common class among these instances. The basic distance-weighted kNN algorithm is presented in Algorithm 10.

6.3.2 Privacy Preserving kNN Classification

The privacy preserving kNN classification problem is described as follows. There exist $n$ private databases $D_1$, $D_2$, ..., $D_n$ distributed at $n$ different nodes. We assume that all the databases have the same schema, i.e. data is horizontally partitioned. These $n$ nodes want to train a kNN classifier on the union of their databases. However, each node would like to reveal as little information as possible about its data to the other nodes during the construction of the classifier (the training phase) and the classification of a new query instance (the test phase).

To solve the privacy preserving kNN classification problem, we need to adapt the basic distance weighted kNN classification algorithm (Algorithm 10) to work in a distributed setting in a privacy preserving manner. One way to do this is to break the kNN classification problem into two sub-problems and solve each of them in a distributed, privacy preserving manner.

A $k$NN classifier views instances as points in a $|A|$-dimensional space, where $|A|$ is the number of attributes of each instance. Given a query instance (point), a $k$NN classifier uses the $k$ nearest neighbors of the query to classify it. In a distributed setting, the $k$ nearest
neighbors of a query point could be distributed among the \( n \) nodes. Thus, each node may have some points in its database which are among the \( k \) nearest neighbors of the query point. So, for a node to calculate its local classification of the query point, it has to first determine which points in its database are among the \( k \) nearest neighbors of this query point.

Let the distance between the query and its \( k^{th} \) nearest neighbor be \( \Delta \). All points which are closer than \( \Delta \) from the query are among the \( k \) nearest neighbors of the query. If a node knows \( \Delta \), then it can determine which points in its database are among the \( k \) nearest neighbors of the query. Thus, the first sub-problem the nodes have to solve is to determine \( \Delta \), which is the distance of the query to its \( k^{th} \) nearest neighbor.

Once a node knows which points in its database are among the \( k \) nearest neighbors of the query, it can calculate its classification of the query instance. Then, the nodes need to combine their local classifications, in a privacy preserving manner, to compute the global classification of the query point over the \( n \) private databases.

Thus we can divide the kNN classification problem into the above two sub-problems. We call the first sub-problem “Privacy preserving nearest neighbor selection” and the second sub-problem “Privacy Preserving Classification”. Let us denote the query to be classified by \( x \). We summarize the two sub-problems are follows:

1. **Privacy preserving nearest neighbor selection:** In this step, the \( n \) databases work together to determine the distance of the \( k^{th} \) nearest point (instance) to \( x \) among all the points in the union of their databases. Knowing the distance of the \( k^{th} \) nearest neighbor helps each node identify all points in its database which are among the \( k \) nearest neighbors of \( x \).

2. **Privacy Preserving Classification:** With the knowledge of which points in its local database which are among the \( k \) nearest neighbor of \( x \), each node can contribute to the global classification in two steps. First, it calculates the local classification of \( x \). Second, with the local classification of \( x \) as input, all nodes engage in a privacy preserving computation of the global classification of \( x \).
6.3.3 Analysis of the Model

The central thesis of our privacy preserving kNN classification model is to divide the problem of classification into two steps, and to ensure that each step is accomplished in a privacy preserving manner. Concretely, given a query instance, a node has to first decide which of its data points are among the $k$ nearest neighbors of the query; these are the points which determine the classification of the query. Our model encourages each node to first calculate the distance of the query to its $k^{th}$ nearest neighbor, using only the distances of the query to the node's data points as input. Note that we utilize only the distances of points from the query instance and not the actual points in a node's database. Thus, whatever information is leaked is only about the distance of a point from the query instance, and never the actual co-ordinates of an instance itself. This is an inherent strength of our model and guarantees that the actual coordinates of a point in a node’s database is never revealed to other nodes.

In the second step, each node can use the information to determine its local classification of the query point, and combine these local classifications in a privacy preserving manner to determine the global classification. This two step division ensures that a node does not have to reveal its data points in order to determine if a point in its database is among the $k$ nearest neighbors of the query.

It is important to note that if executed naively, the above steps can violate the privacy requirements of the individual databases. Given a query point $x$, in the first step, we should ensure that the instances in a database are not revealed to other databases during the computation of the distance of the $k^{th}$ nearest neighbor to $x$. In the second step, we should ensure that the local classification of each database is not revealed to other databases during the computation of the global classification.

Before describing concrete algorithms for privacy preserving $k$NN classification, we emphasize the fact that the model suggested is one possible model for the construction of a privacy preserving $k$NN classifier, and, to the best of our knowledge, it is the first such model to be suggested. Other models may be possible. It would be interesting to study what characteristics determine the best model under different privacy requirements.
6.4 The Algorithm

Based on our privacy preserving kNN classification model, the problem of mining multiple private databases using a kNN classifier can be addressed in two phases. In the privacy preserving nearest neighbor selection phase, each node identifies the points in its database which are among the $k$ nearest neighbors of the query. In the privacy preserving classification phase, the nodes compute their local classifications of the query and combine them to calculate the global classification. We now present algorithms for each phase that help nodes achieve their objectives in each phase in a privacy preserving manner.

6.4.1 Privacy Preserving Nearest Neighbor Selection

Privacy preserving nearest neighbor selection: Let $x$ be the query instance to be classified. Assume that $y_1, y_2, \ldots, y_k$ are the $k$ nearest instances to $x$ in that order, and let $d(y_1), d(y_2), \ldots, d(y_k)$ be the distances of the $k$ nearest points to $x$. In order to determine the points in their database which are among the $k$ nearest neighbors of $x$, the nodes need to determine $d(y_k)$. The $n$ databases engage in a secure computation of this value, which is the distance of the $k^{th}$ nearest neighbor in $D_1 \cup D_2 \cup \ldots \cup D_n$ to $x$.

The algorithm to compute $d(y_k)$ in a privacy preserving manner is as follows. Each node calculates the distance of every point in its database from $x$, and selects the $k$ smallest distances. With these $k$ smallest distances as input, the nodes participate in a privacy preserving computation of the $k$ smallest values among all their inputs. The $k$ smallest values so determined will be the distances of the $k$ nearest points to $x$ among all the points in the $n$ databases. Thus, the $k^{th}$ smallest value will be the distance of the $k^{th}$ nearest neighbor to $x$ among all the points in the union of the $n$ databases.

The crucial step in the above algorithm is the privacy preserving computation of the $k$ smallest values among all the nodes, with each node contributing $k$ values as input to the computation. This can be accomplished by using the “privacy preserving TopK selection” algorithm ($PP-TopK$) described in Chapter 5. The $PP-TopK$ algorithm determines the “top k” values among $n$ distributed nodes in a privacy preserving manner, with each node providing any number of values as input to the algorithm; the “top k” values are defined
by a linear order among all the values held by the nodes. In our case, the linear order is induced on the values held by the nodes by the “less than” relation. We note that it suffices for each node to provide \( k \) values as input to the algorithm, since the number of values in the output is at exactly \( k \).

After the execution of the protocol, the global vector contains, with high probability, the \( k \) smallest distances to the query instance among all the nodes. Knowing this value, the nodes can proceed to execute the next step in the protocol, namely, \emph{privacy preserving classification}.

### 6.4.2 Privacy Preserving Classification

**Privacy Preserving Classification:** Given the \( k^{th} \) smallest distance \( d(y_k) \), each node can determine the points in its database which are within this distance from \( x \), and compute its local contribution to the classification of \( x \) as a function of these points. The global classification of \( x \) is a function of the \( n \) local classifications of \( x \). The nodes engage in a secure computation of the global classification of \( x \), with each node providing as input its local contribution to this computation. The global classification of \( x \) is known to every node at the end of the computation.

Recall that the \( k\text{NN} \) classifier works as follows (Algorithm 10): The classification of the point \( x \) is determined by its \( k \) nearest neighbors. Each of these neighbors effectively “votes” in favor of its own class, the strength of the vote being determined by its distance to \( x \). We add all these votes and select the class that has the highest vote to be the classification of \( x \).

Thus the local classification of \( x \) that is determined by each node is just the class vector of \( v \) votes, assuming that the number of classes is \( v \). The \( i^{th} \) element of this vector is the amount of vote the \( i^{th} \) class received from the points in this node’s database which are among the \( k \) nearest neighbors of \( x \). However, the class with the local majority at this node may not be the same as the class with the global majority of the votes. In order to classify \( x \), we need a way to determine the class with the global majority of the votes. Our solution to this problem is for each node to compute its local classification vector; and then
to participate in a privacy preserving term-wise addition of these local classification vectors to determine the global classification vector. Once each node knows the global classification vector, it can find the class with the global majority of the vote by determining the index of the maximum value in the global classification vector. Thus the nodes can determine the classification of x without disclosing its local classifications.

In the above step, one may use any privacy preserving addition protocol. In our prototype, we use the privacy preserving addition protocol, suggested in [21]. Assume that there are n nodes, each having a value vi. The nodes arrange themselves into a ring. The first node selects a random number and transmits this to its successor. From this point on, every node adds its value to the value it receives and passes it on. When the first node gets back a value, it adds the difference between the initial random number and its own value to this value to determine the sum of the n values. It then broadcasts the sum to all the nodes. This is the simplest privacy preserving summation protocol we are aware of. For improvements to this protocol to make it robust against collusions, and for a proof of privacy, please refer to [21].

We summarize the privacy preserving protocol for addition of local classification vectors in Algorithm 11.

**Algorithm 11 Privacy Preserving Addition of local classification vectors**

**Input:** The local classification vectors lcv_i of each node. **Output:** A global classification vector gcv = \sum_{i=1}^{n} lcv_i

- The nodes are arranged in a ring, and a random starting node is chosen.
- The starting node initializes a global classification vector (gcv) to a random vector rv, and sends it to its successor.
- Every node, upon receiving a vector vi from its predecessor, transmits lcv_i + vi to its successor.
- When the starting node receives a vector vi from its predecessor, it broadcasts (vi - rv + lcv_i) to all the nodes.
6.4.3 Privacy Preserving kNN Classification

Having presented algorithms for the two sub-problems in our model, we now show how to integrate the two solutions to build a privacy preserving kNN (PP-kNN) classifier:

1. Given an instance to be classified $x$, each node computes the distance of every point in its database from $x$, and selects the $k$ smallest distances. These $k$ values are stored in a local distance vector.

2. The $n$ nodes participate in the privacy preserving top-k selection (Algorithm 9), with inputs as their local distance vectors. At the termination of the algorithm, each node knows the distances of the $k$ nearest neighbors to $x$ in the union of their databases.

3. Each node can determine the distance of the $k^{th}$ nearest neighbor to $x$ from the information above. We call this distance $\Delta$.

4. We assume that there are $v$ classes. Each node calculates a local classification vector ($lcv$) as follows:

   $lcv(i) = \sum_y w(d(x, y)) \delta(f(y), i) \cdot [d(x, y) \leq \Delta]$, for all points $y$ in its database, where $w$ is a normalized weight function, $d(x, y)$ is the distance between $x$ and $y$, $\delta(a, b) = 1$ if $a = b$ and 0 otherwise, $[p]$ for a predicate $p$ evaluates to 1 if the predicate is true, and evaluates to 0 otherwise.

5. The nodes use the privacy preserving addition protocol (Algorithm 11) to do an element-wise addition of their local classification vectors to calculate the global classification vector $gcv$: $gcv(i) = \sum_{j=1}^{n} lcv_j(i)$.

6. Each node assigns the classification of $x$ as

   $\text{classification}(x) \leftarrow \arg \max_{i \in V} gcv(i)$.

6.5 Analysis

In this section, we analyze the privacy preserving kNN algorithm using the three performance metrics — relative accuracy, efficiency and privacy.
6.5.1 Accuracy

The relative accuracy of our privacy preserving algorithm is determined by the accuracies of its two steps.

The accuracy of the privacy preserving nearest neighbor selection step is determined by the accuracy of the $PP - TopK$ algorithm. As $PP - TopK$ is a randomized algorithm, we can only present probabilistic guarantees on its accuracy. Per our analysis in Section 5.4, the protocol produces the $k$ nearest distances with a probability $(1 - P_0 r^2 d^{r-1}/2)^k$. For a fixed $k$, the above probability is monotonically increasing with an increasing value of $r$, i.e., it is more likely that the final result is correct if we execute the protocol for a larger number of rounds. For a fixed $k$, we also note that the probability of the output being correct increases with decreasing $p_0$ and decreasing $d$.

The second step, privacy preserving classification, is deterministic and provably accurate. This is because its accuracy depends on the accuracy of the privacy preserving addition protocol, which was proved to be accurate in [21]. Thus, the relative accuracy of our protocol is determined wholly by the accuracy of the first step, privacy preserving nearest neighbor selection.

The relative accuracy of the privacy preserving classification algorithm is best measured by extensive experiments. In these experiments, we train a privacy preserving classifier and a non-privacy preserving classifier on the same training sets; and then test the performance of each on the same test sets. We present a large number of experimental results in Section 6.6 to show that the accuracy of our algorithm is very close to the accuracy obtained by a distributed $k$NN classifier.

6.5.2 Efficiency

An important design goal for privacy preserving classification algorithms is their efficiency. We would ideally like the complexity of the privacy preserving classification algorithm to be of the same order as the complexity of a non-privacy preserving classification algorithm. There are two main considerations when we evaluate the efficiency of a privacy preserving classification algorithm: computational complexity and communication complexity. Ideally,
we would want both these complexities to be of the same order as that of a non-privacy preserving, distributed classification algorithm. We would further like to have a choice between optimizing the algorithm for efficiency and optimizing the algorithm for privacy. Our privacy preserving $k$NN classification algorithm has both these important properties.

If the $PP-TopK$ algorithm is executed for $s$ number of rounds, then the probability that the global vector contains the correct output is at least $(1 - P_0^s * r^{(s-1)/2})^k$ (Section 4.1). If we want the final output to be correct with a probability greater than $(1 - \epsilon)$, for some $\epsilon$ between 0 and 1, then the number of rounds that we have to run the protocol to achieve this accuracy is asymptotic with respect to $k$, i.e., we can pick a value of $s$ (the number of rounds) such that for any $k$, we achieve a correct final output with a probability greater than $(1 - \epsilon)$.

Thus the communication complexity of the $PP-kNN$ algorithm is $\Theta(n)$, here $n$ is the number of nodes involved in the classification. This implies that the communication complexity of the privacy preserving nearest neighbor selection step is $\Theta(n)$.

The communication complexity of the privacy preserving classification step is $2 * n$, which is exactly the communication complexity of the privacy preserving addition protocol. The computational complexity of $PP-kNN$ is also of the same order as that of an ordinary, distributed $k$NN classifier, as there is no cryptography involved in any of the computations. Thus our $PP-kNN$ algorithm is easily able to accommodate a large number of nodes as well as large databases at every node, and scales well along these dimensions.

6.5.3 Privacy

Privacy breaches can be of two different types in our protocol - those that occur due to the $k$NN classification model and those that occur due to our implementation of the privacy preserving $k$NN algorithm. We discuss both of these below.

In our privacy preserving $k$NN classifier model, we divide $k$NN-classification into two steps - privacy preserving nearest neighbor selection and privacy preserving classification (section 3). Information about the data held by individual nodes can be compromised just by following this model, even the individual steps in the model are executed in a perfectly
secure way. To see why this is true, consider the case when all the \( k \) nearest neighbors of a query are in the database of a single node \( n \). In this case, after the execution of the privacy preserving nearest neighbor selection step, \( n \) can infer that all the \( k \) nearest neighbors of the query are within its database, in other words, there are no data points in any other node's database that are among the \( k \) nearest neighbors of the query.

The above breach of privacy indicates that care is required in the design of the \( k \)NN classification algorithm. We can overcome the probability of the above privacy breach by increasing \( k \), the number of neighbors of the query node that we consider. To see why this is the case, consider the case when \( k = |D_1 \cup D_2 \cup \ldots \cup D_n| \) - we are considering all the points in all the database's to classify the query instance. Thus the result of the nearest neighbor selection step will include all the points in all the databases. In fact, if we use the above classifier, in which all points in all the databases are considered when making a classification, then we can do away with the privacy preserving nearest neighbor selection step in our model. This is because every point in every node is by definition one of the \( k \) nearest neighbors of every query instance. This change increases the privacy preserving nature of the protocol, however, we pay a price in the accuracy of the \( k \)NN classifier constructed, as we demonstrate in our experiments, and also in efficiency when each database has a large number of points.

Privacy breaches may also occur due to the algorithms chosen in implementing the two steps of our model. The privacy preserving nearest neighbor selection step uses the PP-TopK algorithm. As we have discussed earlier, we utilize only the distances of points from the query instance and not the actual points in a node's database. Thus, the information leaked in this step is only about the distance of a point from the query instance, and never the actual co-ordinates of an instance itself. This is an inherent strength of our model and guarantees that the actual coordinates of a point in a node's database is never revealed to other nodes.

The PP-TopK protocol also guarantees that with a high probability, the inputs of any node are not revealed to other nodes during the execution of the protocol. In our application, this means that with a high probability, other nodes do not learn about the distances of a
node’s k nearest points to the query point. This is a subtle but important requirement for privacy. To see why this is the case, consider a two dimensional space and a protocol in which every node learns about the distances of other node’s k nearest points to a query instance. In two dimensional space, a point can be uniquely identified if we know its distance from two known points. Thus, a dishonest node might ask queries in such a way as to find out the distances of a point belonging to some other node from these query instances, and might thus be able to infer the co-ordinates of the point belonging to the other node. We show in our experiments that the probability that a node leaks information about its distance values during the PP – TopK algorithm is very low.

The privacy preserving classification step uses the privacy preserving addition protocol and some local computation at the nodes. Thus its privacy is determined by the privacy of the addition protocol. In [21], it has been proved that this protocol is privacy preserving. Thus, the same holds true for our privacy preserving classification step.

6.6 Experimental Evaluation

In this section, we evaluate the privacy preserving kNN classification algorithm in terms of its relative accuracy as compared to a distributed, non-privacy preserving kNN classification algorithm. We conduct experiments to demonstrate the sensitivity of our algorithm to its various parameters.

6.6.1 Experiment Setup

We use three publicly available datasets in our experiments. The first dataset, GLASS [34], contains data regarding various physical characteristics of different types of glass. The classification problem is to identify the type of glass from its physical characteristics. The study of classification of types of glass was motivated by criminological investigation - at the scene of a crime, the glass left behind can be used as evidence if it is correctly identified. This data set contains 214 instances belonging to 7 different classes, with each instance having 9 attributes. The second dataset, PIMA [79], is a medical dataset used for diagnostic purposes - for predicting whether a patient shows signs of diabetes given data like the 2-hour serum insulin concentration and Body Mass Index. This dataset contains 768 instances belonging
to 8 different classes, with each instance having 8 different attributes. The third dataset, ABALONE [90], was used to predict the age of abalone from its physical characteristics. This data set contains 4177 instances belonging to 29 different classes, with each instance having 8 attributes.

Table 5: $k$NN Classification: Experiment Parameters

<table>
<thead>
<tr>
<th>Param.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td># of nodes in the system</td>
</tr>
<tr>
<td>$k$</td>
<td>parameter in $k$NN classification</td>
</tr>
<tr>
<td>$p_0$</td>
<td>initial randomization prob. used in $PP-TopK$</td>
</tr>
<tr>
<td>$d$</td>
<td>dampening factor for randomization prob. used in $PP-TopK$ algorithm</td>
</tr>
<tr>
<td>$r$</td>
<td>Number of rounds for which $PP-TopK$ algorithm is executed</td>
</tr>
</tbody>
</table>

In each experiment, we performed 100 separate runs on each different dataset. In each run, we randomly partitioned the data into two parts - a training set containing $\frac{3}{4}$ of the data and a test set containing $\frac{1}{4}$ of the data. We trained the classifier on the training set and tested it on the instances in the test set. The results reported are averaged over the 100 runs of each experiment.

We summarize the notation we use to describe the experimental results in Table 5.

6.6.2 Relative Accuracy

Figure 45: Accuracy Comparison of Privacy Preserving Classifier and Regular Classifier

In this experiment, we compare the accuracy of privacy preserving $k$NN classification against a distributed $k$NN classifier. In these experiments, we run the nearest neighbor selection step for one round, with the initial probability $p_0 = 1$ and the randomization
factor $d = 0.5$. Thus, with these parameter values, our algorithm is very efficient - its communication complexity is only 3 times that of an ordinary, distributed $k$NN classifier. We would like to determine the accuracy of this classifier which is optimized for efficiency.

We notice from Figure 45 that although we have chosen very conservative settings for our privacy preserving classifier - we run our nearest neighbor selection algorithm for only one round, the accuracy obtained by our classifier is still very high as compared to an ordinary classifier. Note that in some of the cases, the accuracy of the privacy preserving classifier is actually higher than that of a distributed classifier. This is not a contradiction, because an incorrect result of the nearest neighbor selection step could actually produce a value of “$k$” for which the $k$NN algorithm performs better.

Thus, this experiment indicates that our privacy preserving algorithm matches the accuracy of a distributed, non-privacy preserving $k$NN classifier without sacrificing its efficiency. In the next few experiments, we try to reduce the difference in accuracy of the two classifiers, at the cost of increasing the communication complexity of the privacy preserving classifier.

6.6.3 Effect of Number of Rounds

![Effect of number of rounds on Accuracy for Relative data set](image)

(a) Effect of number of rounds on Accuracy for Relative data set

![Effect of number of rounds on Accuracy for Glass data set](image)

(b) Effect of number of rounds on Accuracy for Glass data set

![Effect of number of rounds on Accuracy for Pima data set](image)

(c) Effect of number of rounds on Accuracy for Pima data set

**Figure 46**: Relative Accuracy with Varying Number of Rounds in Nearest Neighbor Selection

Of the two steps in our privacy preserving $k$NN classifier model, the first step, nearest neighbor selection, is probabilistic in nature while the second step, classification, is deterministic and provably accurate. Thus, the overall accuracy of our classifier is determined
by the accuracy of the nearest neighbor selection step. The accuracy of the nearest neighbor selection step is determined by the accuracy of the \textit{PP – TopK} algorithm. We recall that the accuracy of this algorithm is a function of three parameters - \(p_0\), the initialization probability, \(d\), the randomization factor and \(r\), the number of rounds the protocol is executed. From our discussion of this algorithm, we know that the accuracy of the \textit{PP – TopK} protocol increases when we increase \(r\), or decrease either \(p_0\) or \(d\). In this experiment, we verify whether the accuracy of the privacy preserving \(k\)NN classifier approaches that of the distributed \(k\)NN classifier when we run the \textit{PP – TopK} protocol for a larger number of rounds. To do this, we measure the absolute value of the difference between the accuracies of the two classifiers when trained and tested on the same data. The results of our experiments are presented in Figure 46.

We note from Figure 46 that we are able to make the privacy preserving classifier as accurate as an ordinary classifier by running the \textit{PP – TopK} algorithm for a larger number of rounds. This is because the accuracy of the \textit{PP – TopK} protocol increases if we run it for a larger number of rounds; thus, with a high probability, the value of “\(k\)” determined by the nearest neighbor selection step will be the same as the “\(k\)” used by the ordinary algorithm.

6.6.4 Privacy Characteristics

![Figure 47: Loss of Privacy in Nearest Neighbor Selection](image)

In this section, we measure the amount of information revealed by a node to its successor during the execution of the \textit{PP – TopK} algorithm. In this experiment, we run the protocol for \(r = 2\) rounds, with \(p_0 = 1\) and \(d = 0.5\). We measure the probability that a node is able
to correctly identify a value in the global vector it received from its predecessor as belonging to its predecessor. We present these results in Figure 47. From the figure, we note that for all values of \( n \), there is a large range of \( k \) such that the probability of a node revealing information to its successor is less than half. With very large values of \( k \) however, a node has a higher probability of inserting its values in the global vector and this increases its chances of revealing its values to its successor.

Our experiments indicate that even if we run the algorithm for a larger number of rounds, and use a range of values for the randomization factor \( d \), the probability that a node reveals its values to its successor is still very low. However, space restrictions prevent us from presenting the graphs for these results.

6.7 Summary

In this chapter, we have showed how we can aggregate feedback in a privacy preserving way using the primitive privacy preserving protocols through the example of \( k \)NN based reputation computation. This is the first work to show how \( k \)NN classification can be achieved in a privacy preserving manner using a decentralized network protocol. We developed a general model for \( k \)NN classification and presented algorithms for realizing this model. We have laid out the set of requirements that all privacy preserving classifiers should strive to achieve and have analyzed how well our algorithm achieves these requirements. Our algorithm is characterized by its ability to achieve a balance between three important performance metrics: relative accuracy, efficiency, and privacy. This enables our privacy preserving \( k \)NN classification framework to be applied to a variety of problem settings and meet different optimization criteria.
CHAPTER VII

RELATED WORK

Reputation and trust related research stands at the crossroads of several distinct research communities, most notably information systems and information security as well as economics and sociology. We first review the general concept of trust and give a brief overview of the area of trust management research in distributed security. We then discuss related works in reputation based trust research in e-commerce communities and P2P systems. We also briefly review collaborative filtering based recommender systems that are closely related to reputation systems. Finally, we discuss privacy related work in data management area.

7.1 Trust Management in Distributed Systems Security

Trust is a fundamental concept in computer security yet remains an ambiguous term. There are numerous notions of trust and different kinds of trust often satisfy different properties. One of the first works that tried to give a formal treatment of trust was that of Marsh [59]. The model is based on social models of interactions and real world social properties of trust. So the model is rather complex and hard to implement. McKnight et al. [64] proposed a classification of various meanings of trust and Swarup et al. [84] further grouped them into three broad categories: System Trust, Entity Trust and Dispositional Trust. System trust refers to trust that is based on the perceived properties or reliance on the system or institution within which that trust exists. Entity Trust includes all trust relationships between two or more entities. Dispositional Trust describes the general trusting attitude of an entity.

Most of the security literatures focus on Entity Trust. A number of literatures [84, 1] characterize Entity trust as Subjective, Non-transitive, Context specific, Dynamic and non-monotonic. Our reputation based trust framework is also referring to the entity trust or
interpersonal trust among peers in a community.

Trust plays an important role in distributed systems security, in particular, authorization mechanisms. With a formalized trust model, a secure system will be able to answer the question why and how much is a user U trusted to perform actions correctly by stating the parameters and contexts on which it has built its trust assumption about that user. The essential question of authorization is Does a user U have the rights to perform action A on resource R.

Trust relationships in secure systems were first explored in a distributed authentication perspective. Yahalom et al. [96, 13] developed an effective formalism for explicit expressions of trust relations between entities involved in authentication protocols. A formalism for expressing trust relations is presented along with an algorithm for deriving trust relations from recommendations.

There is a growing body of research on trust management aiming at finding a more flexible, more distributed approach to authorization. In distributed systems security, participants use identity certificates to identify each other; they need to trust other participants to issue good certificates and to recommend reliable third parties for accomplishing specific tasks, including signing certificates. Traditional authorization mechanisms fail to provide powerful and robust tools for handling security at the scale necessary for today's Internet [15]. The trust-management approach to distributed system security was developed as an answer to the inadequacy of traditional authorization mechanisms.

Traditionally every application implemented its own mechanisms for specifying access policy, checking compliance and binding user authentication to authorization to perform security-critical operations. The main aim in the development of the trust management system was to produce an off-the-shelf security module that could be integrated into any application. The module allows each application to define application-specific policies and credentials.

The solution that the trust management system proposes is to bind a set of specific keys directly to authorization to perform a specific task, known as a credential. Instead of the traditional question, the fundamental question that the trust management system
now asks is Does the set C of credentials prove that the request r complies with the local security policy P?. Trust management engine takes (r,C,P) as input, outputs a decision about whether compliance with policy has been proven.

Blaze et al. [15] described the main existing trust-management engines, including PolicyMaker [16] and KeyNote [14]. PolicyMaker was the first tool for processing signed requests that embodied the trust-management principles developed by Blaze et al. [16]. The main technical contribution of the PolicyMaker is a notion of proof of compliance that is fully specified and analyzed. KeyNote [14] can be considered a successor to PolicyMaker with extra design goals including easier integration for application and standardization.

Automated trust negotiation [104, 103, 105] is a new approach to access control and authentication for the open, flexible systems formed by sets of organizations that must dynamically form coalitions and work together to respond to unforeseen needs and opportunities. Automated trust negotiation enables open computing by assigning an access control policy to each resource that is to be made accessible to "outsiders"; an attempt to access the resource triggers a trust negotiation, consisting of the iterative, bilateral disclosure of digital credentials and related information.

### 7.2 Reputation and Trust in Electronic Communities

Trust plays important roles in wide-area distributed agent systems. It is an indication of the willingness of agents to associate and cooperate. But how do agents develop trust in each other? Swarup et al. [84] examined formal models of trust propagation in the name-key binding problem context and also described some of their limitations. Most recently, Richard et al. [74] proposed a web of trust for semantic web in which each user maintains trusts in a small number of other users and the trust is propagated through the network. Guha et al, also proposed a framework for propagating trust and distrust given the initial trust relationships [37].

Trust production and trust propagation cannot be investigated separately. We need computation models of trust that enable complex trust decision rules to be established dynamically for both direct trust and indirect trust from propagation. Reputation is one
way for an entity to establish trust in another entity. We first review reputation based trust in e-commerce and agent systems and then review recent reputation systems proposed specifically for peer-to-peer (P2P) systems.

7.2.1 Reputation and Trust in E-Commerce and Agent Communities

Dellarocas [27] provides a working survey for research in game theory and economics on the topic of reputation. Mui et al. [66] also give a review summarizing existing works on reputation across diverse disciplines including distributed artificial intelligence, economics, and evolutionary biology.

The game theory based research [53, 42, 45] lays the foundation for online reputation systems research and provides interesting insight into the complex behavioral dynamics. Most of the game theoretic models assume that stage game outcomes are publicly observed. Online feedback mechanisms, in contrast, rely on private (pair-wise) and subjective ratings of stage game outcomes. This introduces two important considerations, the incentive for providing feedback and the credibility or the truthfulness of the feedback [27].

We review below a number of representative reputation systems and mechanisms that are recently proposed for online environments and agent systems and point out the limitations and the differences from our framework as appropriate.

Abdul-Rahman et al. [1] proposed a model for supporting trust in virtual communities, based on direct experiences and reputation. They introduced the semantic distance of the ratings. However, there are certain aspects of their model that are ad-hoc, such as the four trust degrees and fixed weightings assigned to the feedback. Pujol et al. [69] applied network flow techniques and proposed a generalized algorithm that extracts the reputation in a general class of social networks.

Josang et al. [47, 48] developed and evaluated the beta reputation system for electronic markets based on $\beta$ distribution by modeling reputation as posterior probability given a sequence of experiences. Among other things, they showed that a market with limited duration rather than infinite longevity of transaction feedback provides the best condition.

Sabater et al. [76] proposed Regret system and showed how social network analysis can
be used in the reputation system. Sen et al. [78] proposed a word-of-mouth reputation algorithm to select service providers. Their focus is on allowing querying agent to select one of the high-performance service providers with a minimum probabilistic guarantee.

Yu et al. [97] developed an approach for social reputation management and their model combines agents' belief ratings using combination schemes similar to certainty factors. The reputation ratings are propagated through neighbors using a gossip protocol. In contrast, we build reputation based on transaction feedbacks as against using word-of-the-mouth testimonies. In particular, they suggest refining personal opinions differently for cooperation and defection in order to achieve 'hard to build, easy to drop' property for the reputation. Inspired by this work, we propose a dependable trust model based on a formal PID model popularly used in control theory [67] as a systematic way to address the oscillating behaviors of peers.

Following up the social mechanism for distributed reputation management, Yu et al. also proposed techniques for detecting deception in reputation propagation and aggregation [99]. They adopted weighted majority techniques to handle noisy ratings. The basic idea is similar to PeerTrust in that it associates a credibility weight with each agent when aggregating their opinions. However, in contrast to PeerTrust, all the weights are initialized to 1 and they only get updated (downgraded) after unsuccessful predictions. In other words, the bad peers only get recognized (having their credibility weights downgraded by other peers) after their dishonest feedback incurs a transaction between a malicious peer and an honest peer. On the other hand, the malicious peers who badmouth good peers may never get recognized as the good peers who get malicious dishonest feedback may not get a chance to interact with other peers. In current PeerTrust framework, the derived similarity automatically reflects the changes in the ratings. However, the deception models they have developed benefited the research in this dissertation. It would be also interesting to explore the possibilities to allow the credibility to be updated by users as in their work.

Zacharia et al. [106] proposed an approach that is an approximation of game-theoretic models and studied the effects of feedback mechanisms on markets with dynamic pricing using simulation modeling.
A few proposals specifically attempted to address the issue of quality of the feedback and the deception behaviors of participating parties. Chen et al. [19] differentiate the ratings by the reputation of raters that is computed based the majority opinions of the rating. Adversaries who submit dishonest feedback can still gain a good reputation as a rater in their method simply by submitting a large number of feedbacks and becoming the majority opinion.

Dellarocas [25] proposed mechanisms to combat two types of cheating behavior when submitting feedback. The basic idea is to detect and filter out exceptions in certain scenarios using cluster-filtering techniques. The technique can be applied into feedback-based reputation systems to filter out the suspicious ratings before the aggregation.

A recent paper by Miller et al. [65] proposes a mechanism, based on budget-balanced payments in exchange for feedback, that provides strict incentives for all agents to tell the truth. This provides yet another approach to the problem of feedback trustworthiness. However, such mechanism is vulnerable to collusion. The development of effective mechanisms for dealing with collusive manipulations of online reputations systems is currently an active area of research.

A few works also attempts to address the issue of dynamic behaviors of entities. Dellarocas [28] has shown that storing feedback information on the most recent time interval is enough; and that summarizing feedback information for more than one window of time interval does not improve the reputation system. It suggests that less information is better in certain conditions, in this case, there are no errors in the feedbacks and that all nodes behave rationally. In the presence of dishonest feedbacks there are bound to be errors in identifying an honest feedback from a dishonest one. Further, our experiments show that the history component helps in stabilizing the system by avoiding transient fluctuations due to transient errors or dishonest feedbacks.

7.2.2 Reputation Systems for P2P Networks

A number of reputation based trust systems have been proposed recently for P2P networks. They differ in a variety of aspects, including application domains, inference methodologies,
and implementation strategies.

Related to the feedback sparsity issue, a few reputation systems use inference schemes to propagate trust through network by assuming certain transitivity of trust relationships such as EigenTrust [49]. Recursively, a user will trust the feedback of the trusted acquaintances of his or her friends, in other words, trust may propagate through the network. These schemes typically involve a high computation cost. On the other hand, Marti [60] suggested limited reputation sharing. While it has benefits in load balancing but it is not feasible when the feedback data is sparse.

A common strategy to combat the dishonest feedback is to use certain metric to weigh the information and opinions collected from other peers. A user is much more likely to trust a credible feedback source or trusted acquaintance than from a stranger. There have been a few classes of metrics that have been used. One common way is to use the peer’s previously determined reputation score to weigh its feedback. Representative works including EigenTrust [49] and limited reputation sharing [60]. We argue it is important to compute a separate credibility score for each peer (either personalized or global) and use this score to weight the feedback.

There have been a few efforts trying to classify the trust schemes based on various dimensions [108, 107, 61]. We begin by discussing a few representative works in the field and, when applicable, examining what components and techniques were used to address the feedback related vulnerabilities that we study.

Aberer et al. [3] are one of the first in proposing a reputation based management system for P2P systems. However, their trust metric simply summarizes the complaints a peer receives and files and is very sensitive to the skewed distribution of the community and misbehaviors of peers.

P2PRep proposed by Cornelli et al. [23] is a P2P protocol where servants can keep track of information about the reputation of other peers and share them with others. Their focus is to provide a protocol complementing existing P2P protocols, as demonstrated on top of Gnutella. XRep [24] extends P2PRep by assigning a reputation value for both peers and resources. However, there are no formalized trust metric and no relevant experimental
results in the paper validating their approach.

Another work is EigenTrust proposed by Kamvar et al. [49] that we have discussed earlier. Their algorithm again focuses on a Gnutella like P2P file sharing network. They based their approach on the notion of transitive trust and do not differentiate recommendation trust (credibility) and service trust (reliability). The system starts with the normalized personal feedback matrix as the initial trust matrix. The trust is then propagated through the whole network and the resulting global reputation trust vector is the Eigenvector of the initial personal feedback matrix. They rely on pre-trusted peers to help minimize the effect of collusion problems. By assuming there are peers in the network that can be pre-trusted. While the algorithm showed promising results against a variety of threat models, we argue that the pre-trusted peers may not be available in all cases and a more general approach is needed. Another shortcoming of their approach is that the implementation of the algorithm is very complex and requires strong coordination and synchronization of peers. They assume there are pre-trusted peers in the network and each peer has to place at least some trust in the pre-trusted peers. This breaks up potential malicious collectives as well as guarantees convergence of the algorithm mathematically. However, it remains a research question how to choose pre-trusted peers in a network. While the algorithm showed promising results against a variety of threat models, we argue that the pre-trusted peers may not be available in all cases. Another shortcoming of their approach is that the implementation of the algorithm is very complex and requires strong coordination and synchronization of peers.

Yu et al. [101] proposed a distributed reputation mechanism for P2P networks. They focus on the reputation management for unstructured networks and they manage the trust relationship through a process of referrals. Some of the techniques to handle noisy ratings are built on top of their previous work [99] that we have discussed earlier.

Limited reputation sharing scheme proposed by Marti et al. [60] uses only limited or no information sharing between nodes. The rating of a peer is the combination of the local statistics and the quorum rating, which takes a weighted average of opinions of other peers. A larger weight is given to the nodes that have a high reputation trust value by behaving well and providing good files. In contrast to global history schemes, the focus is to use
limited sharing of opinions. They studied some interesting effects related to load-balancing and message traffic in the P2P network. A problem with this scheme, as the paper shows, it can be easily defeated by a group of malicious peers who set up front nodes that properly trade only authentic files, but when asked for their opinion of other nodes, only promote malicious nodes in the group. This problem can be alleviated when peers consider their own local statistics and give very small consideration of other peers’ opinions. However, when the input matrix is sparse, majority of the nodes will not have statistics for most of other nodes.

Our work differs from them in a number of ways. First, we take a coherent approach to analyze the trust problems in communities and identify the important trust parameters in addition to the feedback in order to effectively evaluate the trustworthiness of peers and to address various malicious behaviors in a P2P community.

We address the problems of dishonest feedback and sparse feedback by a similarity inference scheme. We address the strategic oscillating behaviors of peers with a dependable trust model. Further, none of these approaches point out and address the privacy issues associated with reputation systems. Finally, we also emphasize on how to implement the solution in P2P network in an efficient and secure manner and present detailed algorithms and experimental evaluation of our approach in a distributed P2P environment.

7.3 Recommender Systems

A related area of research is recommender systems based on collaborative filtering techniques. Adomavicius et al. [5] provides a latest survey of the state-of-the-art in recommender systems. The task in collaborative filtering is to predict the utility of items to a particular user based on a database of user votes from a sample or population of other users. In a typical CF scenario, there is a list of users and a list of items. Each user has a list of items for which the user has a rating. Each item has a list of users who have rating for the item. The rating can be explicitly given by the user as a rating score, or can be implicitly derived, e.g. from the purchase records or web access logs. The task of CF is to find an item likelihood for an active user in two forms, prediction and recommendation. Prediction
is a predicted numerical value expressing the likeliness of a target item. Recommendation is a predicted list of items that the active user will like the most.

Collaborative filtering algorithms are generally classified into two classes: memory-based and model-based. The memory based algorithm predicts the votes of the active user on a target item as a weighted average of the votes given to that item by other users. The weight assigned to each user typically reflects distance, correlation, or similarity between that user and the target user. Typical measures used to assign the weight include Pearson correlation and vector similarity. Memory based algorithms have the advantage of being able to rapidly incorporate the most up-to-date information and relatively accurate prediction, but suffer from poor scalability for large number of users.

The model-based algorithm views the problem as calculating the expected value of a vote from a probabilistic perspective and uses the users’ preferences to learn a model. Typical models include cluster models and Baysian networks. The model can be built off-line over several hours or days. The resulting model is small, fast and essentially as accurate as memory-based methods. However, they are not suitable for environments in which user preference models must be updated rapidly or frequently.

Reputation systems are closely related to the memory based collaborative filtering algorithms. In collaborative filtering, users give ratings about items. So the opinion matrix is equal to the ratings matrix where each cell represents a rating from a user about an item. The task of collaborative filtering is to predict the ratings for the empty cells in the matrix.

A natural approach to estimate the value for a specific item is to aggregate the opinions of many users. In a typical neighborhood based method [41], a subset of users are selected as a set of predictors and a weighted combination of their opinions are computed based on their similarity with the active user. In reputation systems, the problem can be formulated similarly as a matrix of service consumers versus service providers, with each cell representing a user’s rating on a specific service provider. Note that in a P2P setting, a peer can be both service consumer and service provider. The problem is to estimate a trust (reputation) score for specific empty cells (user-service provider pairs) in terms of their service performance.
However, there are also important differences between reputation systems and collaborative filtering based recommender systems. First, recommender systems are concerned with items or products that are fairly static. Second, there is no notion of transaction in recommender systems. Third, there is no notion of trust propagation among the users. In short, reputation systems for P2P communities are far more complex considering the dynamics and potential malicious behaviors of the entities.

S. K. Lam and J. Riedl [54] experimentally studied several types of shilling attacks on collaborative filtering systems. We benefited from their attack designs and modeled and analyzed feedback manipulation attacks on reputation systems and developed counter measures.

There has also been some research that attempted to alleviate the sparsity problem in collaborative filtering. Dimension reduction techniques aim at reducing the dimensionality of the user-product matrix by removing unrepresentative or insignificant users or products to condense the user-product matrix. A simple strategy is to form clusters of users or products and then use these clusters as basic unites to make recommendations. Empirical results have shown that dimension reduction techniques improve recommendation performance in some applications but perform poorly in others [77] as potentially useful information might be lost during the reduction process.

Huang et al. [44] modeled the user-product interactions as a bipartite graph (the nodes are divided into two distinctive sets, namely, user set and product set) and modeled the recommendation problem as association retrieval problem (finding associations between user and product nodes). The main idea is to explore and incorporate transitive associations to alleviate the sparsity problem. Spreading activation algorithms developed in associative retrieval literature were applied to efficiently explore transitive associations. Experimental results indicated that spreading activation based collaborative filtering can achieve significantly better results than stand collaborative filtering that do not take into account transitive associations. However, their framework only deals with systems that have binary transactional data. Most importantly, they did not consider the potential vulnerabilities of the system.
It is worth noting that many techniques that are developed in this dissertation are drawn upon collaborative filtering techniques and are as well applicable to recommender systems.

7.4 Privacy Preserving Data Management

Privacy related problems in databases have been an active and important research area. Research in secure databases, Hippocratic databases and privacy policy driven systems [46, 10, 8] has been focused on enabling access of sensitive information through centralized role-based access control. More recently research has been done in areas such as privacy preserving data mining, privacy preserving query processing on outsourced databases, and privacy preserving information integration.

In privacy preserving data mining [89], the main approach is to use data perturbation techniques to hide precise information in individual data records, as the primary task in data mining is the development of models and patterns about aggregated data. In database outsourcing scenarios, the main technique is to use data partitioning to evaluate obfuscated range queries with minimal information leakage [40, 43]. These techniques may not apply to information integration tasks where precise results are desired.

In the information integration domain, Agrawal et al. [9] introduced the paradigm of minimal information sharing for privacy preserving information integration. Under this paradigm, a few specialized protocols have been proposed, typically in a two party setting, e.g., for finding intersections [9], and kth ranked element [6]. Though still based on cryptographic primitives, they achieve better efficiency than traditional multi-party secure computation methods by allowing minimal information disclosure. As a contrast, our protocol does not require any cryptographic operations. It leverages the multi-party network and utilizes a probabilistic scheme to achieve minimal information disclosure and minimal overhead.

The approach of protecting privacy of distributed sources was first addressed by the construction of decision trees [55]. This work closely followed the traditional secure multiparty computation approach and achieved perfect privacy. A key insight of this paper was to trade off computation and communication costs for accuracy, thereby improving efficiency.
over the generic secure multiparty methods. There has since been work to address association rules in horizontally partitioned data [50], association rules in vertically partitioned data [87], and Naive Bayes classification in horizontally partitioned data [51] and vertically partitioned data [88]. However, to the best of our knowledge, this is the first work to present protocols for the privacy preserving kNN classifier construction.

Another related area is the anonymous network where the requirement is that the identity of a user be masked from an adversary. There have been a number of application specific protocols proposed for anonymous communication, including anonymous messaging (Onion Routing [86]), anonymous web transactions (Crowds [72]), anonymous indexing (Privacy Preserving Indexes [12]) and anonymous peer-to-peer systems (Mutual anonymity protocol [92]). Some of these techniques may be applicable for data integration tasks where parties opt to share their information anonymously. However, anonymity is a less strong requirement than data privacy.
CHAPTER VIII

CONCLUSION AND OPEN ISSUES

We have presented a decentralized and dependable reputation based trust supporting framework that helps establishing trust among participating parties. The basic trust framework includes a trust model with important trust parameters identified and a set of decentralized implementation strategies. We then refined the basic framework and developed models and techniques to make the framework more resilient and dependable. In particular, we focused on feedback aggregation related vulnerabilities, including feedback sparsity, feedback oscillation, and loss of feedback privacy, and presented threat models and countermeasures against them.

In this section, we discuss some remaining questions in the current approach for addressing the risks that we have studied as well as some open issues that this dissertation does not directly address.

As we have discussed earlier, reputation systems are important for fostering trust and minimize risks in two ways. First, they help participants estimate the trustworthiness of others and avoid malicious ones to reduce risk. Second, the very presence of a reputation system creates the expectation of reciprocity or retaliation in future behavior, which in turn creates an incentive for good behavior and discourages malicious behavior. We have studied and verified experimentally that the reputation system helps nodes to avoid malicious nodes and in turn increases the overall transaction success rate in a community. However, it would be interesting to conduct a quantitative study evaluating how much a reputation system actually encourages good behavior and discourages malicious behavior.

The proposed similarity inference scheme counters feedback sparsity by utilizing the transitive associations of similarity among peers. It essentially clusters similar users together based on their feedback and helps them to evaluate each other despite sparse feedback. An
interesting and opposite phenomenon that has been suggested is that when recommendations are generated in a distributed manner with scattering, the quality of the network could improve when clusters are reduced [80, 102]. This corresponds to the intuition that people benefit from knowing others outside their patochial groups. This also suggests an interesting research opportunity that considering dissimilarity as well as similarity in the inference scheme may have unexpected yet potentially good results.

While we have shown the PID based approach effectively handles the oscillating behaviors of peers, it has also been shown [28] that storing feedback information on the most recent time interval is enough; and that summarizing feedback information for more than one window of time interval does not improve the reputation system assuming that there are no errors in the feedbacks and that all nodes behave rationally. This suggests that less information is better in certain conditions. It would be interesting to study other controllers from control theory versus the PID controller and also study the impacts and differences of using less information versus more information in different conditions.

A main motivation for us to develop techniques to preserve feedback privacy in the reputation system is to encourage peers to provide honest (especially if negative) feedback without the fear of being retaliated. We have showed that the proposed protocols can help minimize the loss of feedback privacy in feedback aggregation process. However, it is not clear how much preserving feedback privacy would actually affect the feedback filing behavior of peers. Again, a future quantitative study may provide some insights.

Our work also continues in improving the protocol design. First, we would like to explore the information theory based privacy metrics and incorporate those in our protocol design. Second, we plan to relax the semi-honest model assumption and address the situations where adversaries may not follow the protocol correctly. Finally, we are interested in developing an adaptive protocol based on different privacy requirements and trustworthiness of individual peers.

There are also risks and threats that cannot be fully prevented or detected by current approach. We discuss a few of them and suggest future study of potential corrective and preventive solutions for recovery and survival.
There is so far no mechanism that can completely prevent the attack of peers being compromised. We plan to engage in a study of attacks made via anonymous operations, and develop corrective and preventive methods as a part of trust building and trust management research.

Another risk related to the dynamic behaviors of peers is that malicious peers can easily discard their old identity and adopt a new one through reentry as the whitewashing attack has shown. A dependable reputation system also has to provide certain mechanisms to penalize such behaviors while protecting legitimate new users. Friedman [33] discuss two classes of approaches to this issue: either make it more difficult to change online identities, or structure the community in such a way that exit and reentry with a new identity becomes unprofitable.

The proposed trust building techniques are based on experiences, therefore, a peer that has been consistently reliable can perform an unavoidable one-time attack. Although the proposed trust system utilizes an adaptive time window, it is very hard if not impossible to fully prevent all possible one-time attacks.

Therefore, our work on reputation and trust continues in investigating the above remaining risks and explore mechanisms to make the framework more robust against malicious behaviors.

Finally, we are also interested in applying the techniques to other problem domains such as web spam [38, 39] and web service reputation in semantic web [63].
REFERENCES


VITA

Li Xiong was born and grew up in Wuhan, China. After she received a BS in Computer Science from University of Science and Technology of China, she went to Johns Hopkins University to pursue graduate studies. She left her PhD program after she received an MS in Computer Science and worked several years as a software engineer with companies including SRA International and Internet Security Systems. She returned to school afterwards and is now completing her PhD in College of Computing at Georgia Institute of Technology. Upon finishing her dissertation, she will be joining the faculty at Mathematics and Computer Science department at Emory University. Her general areas of interests are in data and information management, distributed computing, trust and information privacy.