SECURITY AND PRIVACY ISSUES IN SOCIAL INFORMATION-ASSISTED APPLICATION DESIGN

A Dissertation
Submitted to
the Temple University Graduate Board

---

in Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

---

by

Chang, Wei
December, 2016

Examining Committee Members:

Dr. Jie Wu, Dept. of Computer and Information Sciences (Advisor)
Dr. Chiu C. Tan, Dept. of Computer and Information Sciences (Committee chair)
Dr. Wei-Shih Yang, Dept. of Mathematics
Dr. Avinash Srinivasan, Dept. of Computer and Information Sciences
Dr. Saroj K. Biswas, Dept. of Electrical and Computer Engineering (External reader)

ABSTRACT

Security and Privacy Issues in Social Information-Assisted Application Design

by

Chang, Wei

In recent years, social networks and their related theories and applications attract widespread attentions in computer science. Many applications are designed by exploring the social information among users, such as social peer-to-peer systems, mobile cloud, and online recommendation systems. Most of the existing works only focus on how to use social information but ignore the fact that social information itself may cause severe security and privacy problems. In this dissertation, we first present some social information-assisted application systems that we have designed, and then, we present several social information-involved privacy and security risks and their countermeasures.

Generally speaking, the design procedure of any social information-assisted application involves three tasks: publishing, accessing, and using social information. However, all of these tasks contain privacy and security issues.

Social information can be published from a centralized system or a distributed one. For the centralized scheme, the social information is directly published from online social networking systems, such as Facebook or Twitter. However, we found that the data of a social network essentially is a time-evolving graph. Most of the existing approaches fail to preserve users' identity privacy once a malicious attacker has the external knowledge about the victim's time-varying behaviors. For avoiding the new privacy issue, we propose a time-based anonymization scheme. For the distributed social information-sharing scheme, each user's information is propagated from friend to friend's friends, and so on. We design a new scheme to gradually enhance the privacy protection along a propagation path, in the meanwhile, maximally preserve the overall utility of the user's data.

From a data accessing aspect, social information can be used by malicious users for launching new attacks. In this dissertation, we find a friendship-based privacy disclosure attack, and a corresponding defense approach is designed. Location-based service has been widely adopted. In order to preserve location privacy, users usually turn off the corresponding applications when visiting sensitive locations. However, once social relationships are known, attackers are able to infer these hidden locations, which disclose users' location privacy. For preserving the location privacy, we design a fake location-based approach, which efficiently disorders the social-geographic relationships among users.

From the data usage aspect, social information and its related data may come from users. A system may lose functioning if some malicious users inject plenty of fake information. Mobile clouds and Friend Locator are two typical systems, which are vulnerable to the fake information-related attacks. Mobile clouds explore the idle computing resources of surrounding devices by recruiting nearby friends to participate in the same task. However, malicious users may inject wrong friendships information to mess up the system. When visiting a new place, Friend Locator provides navigation services for participators by creating a map based their trajectories. The functioning of the system is based on the trust among participators. Once a user's device is controlled by attackers, all other users may receive wrong navigation. For defending these attacks, we provide different countermeasure.

Keywords: Attack, privacy, security, social information, system design and evaluation.

To my beloved parents

ACKNOWLEDGEMENTS

For the past six years in the Center for Networked Computing (CNC) of Temple University CIS department, I have benefited from the collaborations with brilliant researchers. I always enjoyed the excellent working environment and harmonious atmosphere in this department.

This research could not have been finished without the guidance of my advisory committee members, support from my family, and the help of friends. I would like to express my sincere gratitude to my advisor, Dr. Jie Wu, for his selfless help, immeasurable caring, strict guidance, and patience. He has taught me how to think comprehensively and critically, shown me the importance of persistence and hardworking to success, and provided me with solid supports. I take him as my role model, and I will never forget his guidance.

I would like to thank my committee members, Dr. Chiu C. Tan, Dr. Wei-Shih Yang, Dr. Avinash Srinivasan, and Dr. Saroj K. Biswas, for their sincere help, extensive knowledge, and valuable comments to complete this dissertation.

My special thanks go to Dr. Eugene Kwatny and Ms. Claudia Pine-Simon. It was my pleasure to work with them as a teaching assistant. I deeply appreciate their help and support. I also would like to thank my lab-mates and colleagues for the precious time that we spent together.

Finally, I would like to give my deepest and heartfelt thanks to my beloved parents, Yanli Chen, and Zhixue Chang, for their unconditional love. Thank you for letting me know that I should never settle for less than what I deserve, for giving me the support that I needed to build a dream to chase after, and for believing that I have the strength and capability to realize my dreams.

TABLE OF CONTENTS

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENTS v

LIST OF FIGURES xi

LIST OF TABLES xvii

CHAPTER

1. INTRODUCTION 1

1.1 Motivation and Challenge 1
1.2 Major Contributions 3
1.3 Dissertation Overview 5

2. EXPLORING SOCIAL PREFERENCE IN THE DESIGN OF PEER-TO-PEER SYSTEMS 8

2.1 Introduction 9
2.2 Literature Review 12
2.3 Vimeo Category and User Interests 14
2.4 Overlay Design 16
2.4.1 Feature Coordinate and Feature Distance 16
2.4.2 Channel Subscription and Peer-to-Peer Graph . . . 17
2.4.3 Resource Replication Scheme 18
2.4.4 Overlay Construction 18
2.4.5 Inter-group Link Management 20
2.5 Routing in SocialVoD 21
2.5.1 Coordinate Routing Sequence and Node-disjoint Paths 22
2.5.2 The Selection of Coordinate Routing Sequences . . . 23
2.5.3 Feature Space Routing 25
2.6 Extension 27
2.6.1 Multi-channel Subscriptions 27
2.6.2 Routing for Non-subscribers 29
2.6.3 Pre-fetch for New Videos 30
2.7 Performance Analysis and Evaluation 30

2.7.1 Simulation Setup and Evaluation Metric 31
2.7.2 Simulation Results 32
2.8 Chapter Summary 38

3. EXPLORING SOCIAL MOBILITY IN THE DESIGN OF MOBILE CLOUDS 40

3.1 Introduction 41
3.2 Literature Review 45
3.3 System Model and Problem Formulation 46
3.3.1 System Model 46
3.3.2 Problem Formulation and Challenges 48
3.4 Solution Overview 50
3.4.1 Key Observations 50
3.4.2 Main Idea 51
3.5 Solution Details 52
3.5.1 Contact Frequency and Contact Graph 52
3.5.2 Neighbor Status-based Finishing Time Estimation . 54
3.5.3 Local 1-hop Status-based Workload Allocation . . . 56
3.5.4 Potential Ability-based Workload Allocation 60
3.5.5 Adaptive Scheme 65
3.5.6 Example 68
3.5.7 Further Discussion 69
3.6 Performance Analysis and Evaluation 72
3.6.1 Evaluation Setup and Comparison Metric 73
3.6.2 Evaluation Results 75
3.7 Chapter Summary 82

4. EXPLORING SOCIAL ENCOUNTERS IN THE DESIGN OF COOPERATIVE TRAJECTORY MAP-BASED NAVIGATION 84

4.1 Introduction 84
4.2 Literature Review 86
4.3 Background 88
4.3.1 System Model 88
4.3.2 Existing Noise Cancellation Solutions 90
4.3.3 Challenges 91
4.4 Solution Framework 93
4.4.1 Accessorial Anchor-based Error Reducing Algorithm 95
4.4.2 Anchor-free Error Reducing Algorithm 96
4.5 Algorithm Details 97
4.5.1 Error Model 97
4.5.2 False Positive and False Negative Error Detection . 99
4.5.3 Hypothesis-based Mass-spring Adjustment 101

4.5.4 Historical Error Cancellation 105
4.5.5 Additional Discussion 106
4.6 Routing Application: Friend Locator 107
4.7 Performance Analysis and Evaluation 109
4.7.1 Evaluation Metric 109
4.7.2 Simulation Results 109
4.8 Chapter Summary 114

5. PRIVACY-PRESERVED SOCIAL INFORMATION PUBLISHING FROM A CENTRALIZED SYSTEM 115

5.1 Introduction 116
5.2 Literature Review 120
5.3 System Model and Problem Formulation 121
5.3.1 System Model 121
5.3.2 Attack Model 122
5.3.3 Problem Definition 124
5.4 Overview 125
5.5 Space Compression 127
5.6 Anonymous Group Construction 129
5.7 Realizability and NFS Indistinguishability 133
5.8 Performance Analysis and Evaluation 136
5.9 Chapter Summary 141

6. PRIVACY-PRESERVED SOCIAL INFORMATION PUBLISHING WITHIN A DISTRIBUTED SYSTEM 143

6.1 Introduction 144
6.2 Literature Review 146
6.3 Problem Model and Formulation 148
6.3.1 Basic Model 149
6.3.2 Problem Formulation 151
6.4 Key Observations 155
6.5 Solution Details 156
6.5.1 Bottom-Up Approach 157
6.5.2 Top-Down Approach 160
6.5.3 Forward Looking Approach 162
6.5.4 Example 164
6.6 Security Analysis 166
6.7 Evaluation 169
6.7.1 Simulation Setup and Evaluation Metric 169
6.7.2 Simulation Results 170
6.8 Chapter Summary 172

7. FRIENDSHIP-BASED LOCATION PRIVACY DISCLOSURE IN MOBILE SOCIAL NETWORKS 175

7.1 Introduction 175
7.2 Literature Review 177
7.3 Overview 178
7.4 Proposed Solution 180
7.4.1 User Traveling Alone 180
7.4.2 Users Traveling in a Group 185
7.5 Evaluation 186
7.6 Chapter Summary 189

8. SYBIL ATTACKS IN MOBILE SOCIAL NETWORKS . . . 191

8.1 Introduction 191
8.2 Literature Review 193
8.3 System Model 194
8.4 Signed Network-based Sybil Defense 195
8.4.1 Generation of Signed Network 195
8.4.2 Scheme of Signed Network-based Sybil Defense . . . 196
8.4.3 Security Analysis 199
8.5 Sybil Gateway Breaking 200
8.5.1 Overview 201
8.5.2 Gateway Verification Algorithm 202
8.5.3 Suspicious Edge Selection 203
8.5.4 Attack Edge Detection 204
8.6 Performance Analysis and Evaluation 205
8.6.1 Simulation Metric 205
8.6.2 Simulation Results 206
8.7 Chapter Summary 209

9. WORMHOLE ATTACKS IN COOPERATIVE TRAJECTORY MAP-BASED NAVIGATION 211

9.1 Introduction 211
9.2 Literature Review 215
9.3 Background 216
9.3.1 System Model 216
9.3.2 Trajectory Map Construction 217
9.3.3 Adversary Model 219
9.4 Detection Algorithm 220
9.4.1 Intuition 220
9.4.2 Adversaries Detection and Isolation 221
9.4.3 Trajectory-based Encounter-type Prediction 223
9.5 Technical Details 227

9.5.1 Position Function 227
9.5.2 Possible Trajectory Boundary Function 228
9.5.3 Intersection Function 229
9.5.4 Abnormal Events 230
9.6 Performance Analysis and Evaluation 232
9.6.1 Experimental Setup 232
9.6.2 Evaluation Metric 234
9.6.3 Simulation Results 235
9.7 Further discussion 241
9.8 Chapter Summary 242

10. DISSERTATION CONCLUSION 244

10.1 Summary of Contributions 244
10.2 Future Research 246

PUBLICATIONS 248

BIBLIOGRAPHY 251

LIST OF FIGURES

Figure

1.1	The logic structure of the dissertation's chapters.	6
2.1	Vimeo's feature space structure.	10
2.2	A diagram of the network structure of SocialVoD.	17
2.3	Coordinate routing sequence-based inter-group routing.	21
2.4	The hierarchical community structure in SocialVoD.	25
2.5	Source selection when a user subscribes to multiple channels.	28
2.6	The impact of the buffer size.	33
2.7	The impact of the new content's arriving rate.	33
2.8	The impact of users' watching preferences.	35
2.9	The impact of each peer's total number of links.	35
2.10	The impact of the length of random walks.	36
2.11	Real data-based simulation results.	37
3.1	Workload allocation in social-crowdsourcing.	44
3.2	Contacting graph of example data set.	57
3.3	Single work: case 1.	58
3.4	Single work: case 2.	58
3.5	Single work: case 3.	58
3.6	Single work: case 4.	58
3.7	Single work: case 5.	58

3.8 Work progress comparison. 58

3.9 Special structure: case 1. 59

3.10 Special structure: case 2. 59

3.11 Special structure: case 3. 59

3.12 Special structure: case 4. 59

3.13 Special structure: case 5. 59

3.14 Work progress comparison. 59

3.15 An example about workload allocation in social-crowdsourcing. . . . 68

3.16 Multiple works: case 1. 71

3.17 Multiple works: case 2. 71

3.18 Multiple works: case 3. 71

3.19 Multiple works: case 4. 71

3.20 Multiple works: case 5. 71

3.21 Throughput rate comparison. 71

3.22 The impacts of average accept ratio on processing progress. 74

3.23 Impacts of average accepting ratio on the system throughput rate. . 76

3.24 Impacts of average speed on the system throughput rate. 78

3.25 Impacts of the size of works on the system throughput rate. 79

3.26 Impacts of the average length of inter-contacting time on the system throughput rate. 80

3.27 Impact: the number of sources. 81

3.28 Impact: the number of works. 81

3.29 Impact: arriving density. 81

4.1 The effect of a measurement error. 85

4.2 System model. 88

4.3 The challenge of making adjustments without the auxiliary information. 92

4.4 Error model. 98

4.5 The uncertainty of adjustment direction in false positive cases. . . . 103

4.6 The historical error cancellation. 106

4.7 The effect of measurement error during routing. 108

4.8 An example of simulation used data. 110

4.9 An example of the map construction process. 111

4.10 Time vs. AP/AP-free. 112

4.11 User num. vs. inaccuracy. 112

4.12 AP density vs. inaccuracy. 113

4.13 Error vs. inaccuracy. 113

5.1 An example of evolving social subscription network. Nodes in the same circle indicate the corresponding users subscribed to the same media channel (ex. online discussion groups, or Facebook fans pages). 117

5.2 An example of attacks on ESSN. 118

5.3 Utility loss: whole sequence-based approach vs. single slice-based. . 138

5.4 UL vs. intragroup error. 139

5.5 Impacts of dummy nodes. 139

5.6 Exe. time vs. compression. 140

5.7 Impacts of channel number. 140

5.8 The effects of equivalent channels. 140

5.9 The effects of complementary anonymous groups. 141

6.1 An example of the target problem. Black nodes create a social chain with length 4, dash lines represent the enhanced privacy requirements, and bars stand for the real privacy value of the table. 145

6.2 Generated tables based on requirements with different privacy levels. 147

6.3 The grouping problem with BUA. 160

6.4 Forward looking the future privacy requirement. 161

6.5 The idea of FLA. $\delta_{h+2} < \delta_{h+1} < \delta_h$. 162

6.6 Example (Part I). Part II is in Fig. 6.7. 164

6.7 Example (Part II). The value of tuples $11-24$ in both Figs. (b) and (c) are the same as them in Fig. (a). 166

6.8 Created equivalent classes. 170

6.9 Ave. utility loss. 170

6.10 Ave. privacy. 172

6.11 Utility lost. 172

6.12 Time cost. 172

7.1 An example of friendship-based location privacy disclosure. 176

7.2 Two users travel together. 187

7.3 One user defense. 187

7.4 Error by fake or omission. 187

7.5 Error vs. observing time. 187

7.6 The trajectories and distance histogram between two friends before and after using our proposed method. 188

7.7 Error vs. unreported num. 189

7.8 Error vs. threshold. 189

8.1 System model. 195

8.2 The generation of distrust social profiles. 197

8.3 The computation of trust score. 198

8.4 The computation of distrust score. 199

8.5 The impacts of the number of honest communities. 207

8.6 The impacts of the betweenness threshold. 207

8.7 The impacts of the number of attack edges. 208

8.8 The impacts of the number of distrust relations. 208

8.9 The impacts of the number of Sybil communities. 208

8.10 The impacts of the number of honest communities. 209

9.1 The effect of a wormhole attack. 213

9.2 System model. 217

9.3 Relation between two users' trajectories. 224

9.4 Illustration of encounter-type in X-Y-T space. 226

9.5 Observing time vs. accuracy. 236

9.6 Adversary density vs. accuracy. 237

9.7 User density vs. accuracy. 237

9.8 Sensor range vs. accuracy (overall). 238

9.9 Sensor range vs. accuracy (beginning part). 239

9.10 Observing time vs. false positive and false negative. 239

9.11 Sensor range vs. false positive and false negative. 240

9.12 Adv. density vs. FP & FN. 240

9.13 User. density vs. FP & FN. 240

LIST OF TABLES

Table

2.1 The distribution of Vimeo channels' associated category numbers . . 14

2.2 The Change of Category Interests Over 5 Years (%) 15

2.3 The impact of the push operation for newly generated contents . . . 36

3.1 Common notations 48

4.1 Table of notation 90

4.2 Table of notations for auxiliary functions 99

5.1 The privacy risks analysis 137

5.2 The impacts of similarity metrics (Chapter 5.6) 138

6.1 Notations used in this chapter 149

7.1 Table of notation 178

9.1 Table of notation for TBEP 223

9.2 Table of notation for helper functions 227

CHAPTER 1

INTRODUCTION

In recent years, social networks and their related theories and applications attract widespread attentions in computer science. With the development of clouds and ubiquitous internet accessing, more and more novel and interesting applications are designed by exploiting social information among users. However, using social information is a double-edged sword due to the fact that malicious attackers can also take advantage of social information to launch new attacks. This dissertation systematically expounds various ways to incorporate social information into new applications, and their potential security and privacy risks.

1.1 Motivation and Challenge

There are many types of social information, from individual behavior, mutual social influence, to group behavior. Different social information-assisted applications exploit sociality's different aspects. Typical applications, such as social preference-based peer-to-peer systems, mobile cloud, and mobile social networks, directly use the social features of users. Considering that human being are the physical carriers of the mobile devices and (or) the participants of certain applications, knowing the social information among users surely can improve many applications' quality of services. For some other systems, such as Twitter, crowdsourcing, and cooperative trajectory mapping, their functioning is purely based on users' daily social activities, like an encounter with friends, physically or online. Instead of directly involving social information to the decision-making process, these systems provide new applications for enriching people's social life. One of the research challenges for social information-assisted application design is to discover new social features and apply them to the application design process.

Most of the existing works only focus on the exploration and usage of social information but ignore the fact that, once a sociality is involved, malicious users can take advantage of the social information to launch lots of security or privacy attacks. Clearly, another challenge for creating a new sociality-involved application is to identify and prevent the potential risks. However, since applications may involve diverse social features, distinct service purposes, disparate system architectures, and different algorithm procedures, it is impossible to find out all the potential security and privacy vulnerabilities of an application. In this dissertation, we point out several directions that an attacker may work on, propose new attacks that we have found in these directions, and provide the corresponding countermeasures.

First, social information itself may cause severe security and privacy issues. Before using social information, any application must obtain it in advance, which gives attackers a chance to launch privacy-related attacks on the published social information. Generally speaking, there are two ways to obtain the social information: from a centralized data source, or from individual participators of a distributed system. In this dissertation, we present new privacy disclosure attacks in these two scenarios, and then, provide the corresponding data anonymization schemes. The research challenge of this part is about the trade-off between data utility and data privacy.

Second, social information could also be used by attackers. Location-based social networking service has been widely adopted. Typical applications include Facebook Places, FourSquare, Friends Around, and Buzzd. Through these applications, users are able to not only get the location-based service but also have social interactions, such as comments and chatting. In order to preserve location privacy, users usually turn off the corresponding applications when visiting sensitive locations. However, once a target user's social information, such as the contacting frequency with friends, is available, attackers may be able to infer these sensitive locations since the victim may have visited these places with close friends. To solve the problem, we need to make the balance among security, privacy, and utility.

Third, from the consideration of data usage, using social information does not indicate blind-trust. Social information may be provided not only by regular users but also by the malicious one. The main challenge is how to identify the wrong information. In this dissertation, we use Sybil attack on a social-crowdsourcing system as an example. Instead of verifying each piece of information, we focus on the data source. We create a novel scheme to compute whether a participant is trustable, which can limit the damage of the Sybil nodes on any social information-assisted system. Note that the trust verification mechanism is needed, not only for the social information but also for other data provided by the participators. Friend Locator is an application, which is purely based users' social activity. Based on the reported trajectories, it provides navigation service for friends to find each other. However, if any user's device is controlled by an attack and reports wrong data, the navigation results will become useless.

1.2 Major Contributions

In this dissertation, we carry out the study of social information-assisted application design and its related potential security and privacy risks. Because there are various types of social information and systems, all aspects related to social information-assisted application design, including social information publishing, accessing and usage, may contain security or privacy vulnerabilities. This dissertation, from each aspect, provides typical security/privacy attacks and their defense approaches. The contributions of this dissertation can be summarized as follows:

- For social information-assisted system design, we construct three systems: social P2P, social mobile cloud, and social encounter-based navigation system. For social P2P, we propose a new P2P overlay constructing approach by fully utilizing the social similarity of the users, who subscribe to the same media channel, and exploring the existing category information of media files. For social mobile cloud, also known as social crowdsourcing, we study the workload allocation problem among a set of intermittently connected mobile devices. As for the social encounter-based navigation, we design a social encounter-based systematic error cancellation scheme, which does not require physical anchors.

- In the social information publishing scenario, we study the privacy-preserved data publishing (PPDP) in both centralized system and distributed system. For the centralized PPDP, we find a new privacy attack by exploring the changing pattern of a user's social preference. For the distributed PPDP, we propose a new scheme by gradually enhancing the privacy protection level. The corresponding data anonymization algorithms are designed for each system.

- From a data accessing aspect, once social information is accessed by malicious users, based on the pattern of certain social behavior, they are able to design new attacks. By exploring users' co-appearance frequency with friends, we create a powerful attack to pry into the location privacy of other users, even if they have already turned off their devices. Based on information theory, we make a quantitative analysis of the amount of privacy disclosure, and a new location anonymization algorithm is designed.

- For the scenario of using social information, we consider the authenticity and trustworthy about social data and social actors. We first study the Sybil attacks in mobile social network, where attackers inject fake social information into a system. To limit the impact of Sybil attacks, we provide a distributed trust evaluation mechanism. We also study the wormhole attack in cooperative trajectory mapping system, where malicious social actors report non-existing trajectory data. In order to defend the attack, we propose a scheme to verify the authenticity of the reported data.

1.3 Dissertation Overview

The organization of the remaining chapters of this dissertation is as follows: In the first part of this dissertation, from Chapter 2 to Chapter 4, we use three social information-assisted applications that we have designed to illustrate the benefits of incorporating sociality into application design. In Chapter 2, we propose a new P2P overlay construction scheme based on users' subscription behaviors on modern video-on-demand platforms. In Chapter 3, we design a mobile device-based crowdsourcing system, where the workload allocation decision is made based on users' sociality and the computing capability of their friends' devices. In Chapter 4, we study the impacts of systematic measurement errors on a social information-assisted application, called Friend Locator. We use participants' social encounter records to eliminate the relative errors among their reported trajectory data and provide better navigation results.

From Chapters 5 to 9, we study the security and privacy issues involved with social information-assisted application design. More specifically, we consider the vulnerability from three aspects: (1) Data Publishing, (2) Data Accessing, and (3) Data Usage. These three aspects correspond to the general design procedure of a social information-assisted application: One has to obtain the related social information first before exploring certain social features, and then, determines who can access which kind of data and how to use the data. In this dissertation, we

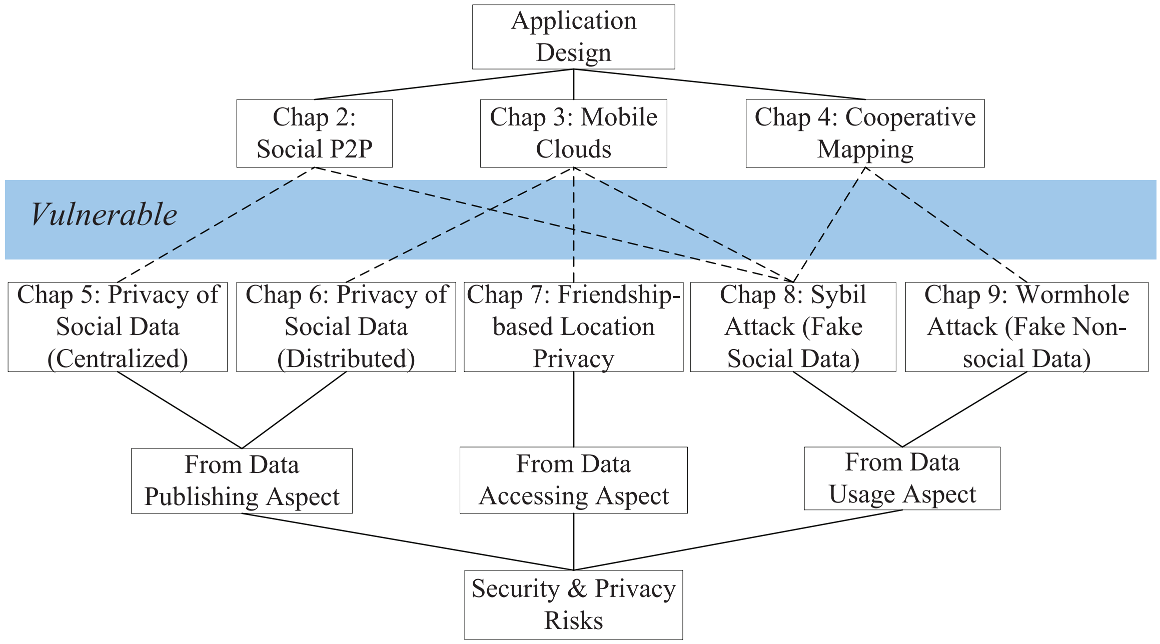


Figure 1.1: The logic structure of the dissertation's chapters.

find that all of these three aspects contain potential security or privacy risks. When designing a new application, the designer should systematically consider the potential vulnerabilities from these three aspects.

The second part of this dissertation, Chapters 5 and 6, considers the security/privacy risks from the aspect of data publishing. Since sociality evolves with time, Chapter 5 shows that the change pattern of a target user's social activity can leak the identity privacy when a single source publishes a serial of sociality-related data. Considering that trust values gradually decrease along a propagation path, Chapter 6 discusses the privacy-preserved data publishing in distributed systems.

Chapters 7 is the third part, which studies the security/privacy risks from the aspect of data accessing. Attackers can also take the advantage of social information to launch new attacks. Chapters 7 provides a friendship-based location disclosure attack, which leaks victim's location privacy.

The fourth part of this dissertation consists of Chapters 8 and 9. Exploiting social knowledge does not mean blind use. Chapter 8 shows that an attacker can subvert a system by maliciously creating a large number of fake user IDs and injecting wrong information into the system through these IDs. For other social information-assisted applications, which do not directly use social data but essentially builds on the actual social relations among users, attacks can also use wrong data to mess them up. Chapter 9 shows the impacts of such attacks on the application of cooperative trajectory mapping. Chapter 10 concludes this dissertation and points out the future research topics.

Note that the applications presented in Chapters 2 to 4 potentially are vulnerable to certain attacks discussed in Chapters 5 to 9. Fig. 1.1 shows the logic connection between each chapter of the dissertation.

CHAPTER 2

EXPLORING SOCIAL PREFERENCE IN THE DESIGN OF PEER-TO-PEER SYSTEMS

Video-on-demand (VoD) service has been explosively growing since its first appearance. For maintaining an acceptable buffering delay, the bandwidth costs have become a huge burden for the service providers. Complementing the conventional client-server architecture with a peer-to-peer system (P2P) can significantly reduce the central server's bandwidth demands. However, the previous works focus on establishing a P2P overlay for each video, producing a high maintenance cost on users.

In order to construct a better P2P overlay, people begin to consider the *social similarity* among users, especially the social behaviors reflecting their common interests. Per-channel-based overlay construction was first introduced by SocialTube [1], which clusters the users subscribed to the same video channels into one P2P overlay. However, the current per-channel overlay structure is not suitable for users developing new watching preferences. Consider that a channel's subscribers tend to watch not only the videos from the channel, but also other videos from similar channels.

In this chapter, we propose a new overlay structure by exploring the existing *social relations of users* and the *similarities of video channels*. Our system creates a hierarchical overlay: subscribers of the same channel build up the low-level overlay (also known as groups), and in the high-level overlay, different groups are connected based on their similarities. Based on the new structure, we propose a routing algorithm for both channel-subscribed and unsubscribed users. Extensive simulation results show the efficiency of our approach.

2.1 Introduction

In the past decade, we have witnessed the enormous growth of video-on-demand (VoD) services, such as Vimeo and YouTube. According to sites [2, 3, 4], Vimeo attracted over 100 million unique visitors per month in 2013. With the development of affordable smartphones and the Internet, more and more VoD platforms have allowed users to create their own media channel and upload videos. For example, Ray Johnson has the most popular personal Youtube channel [5], which has gained 10 million subscribers. Clearly, a tremendous amount of network resources has been used to maintain such a huge supply and demand market. Most of the current VoD systems are based on a client-server (CS) architecture. All videos are stored in central servers; whenever a user tries to watch a video, a query message will be generated and sent to the servers; then, the corresponding video will be downloaded from the servers to the requester. Although the CS architecture is easy to manage, it brings enormous bandwidth costs for the servers' provider and long buffering delays for users.

In order to reduce the bandwidth costs and users' average delays, complementing the CS architecture with a peer-to-peer (P2P) architecture becomes a common approach. In a P2P architecture, users, also known as peers, are both video-resources' suppliers and consumers. Instead of fully depending on central servers, in P2P, peers are able to download videos from other peers, which significantly lightens the servers' burdens. Peers are logically connected at the protocol layer, which builds a network overlay on the top of the underlying physical network [6]. The research on P2P has existed for decades, and the main concern is how to construct such a logical overlay for

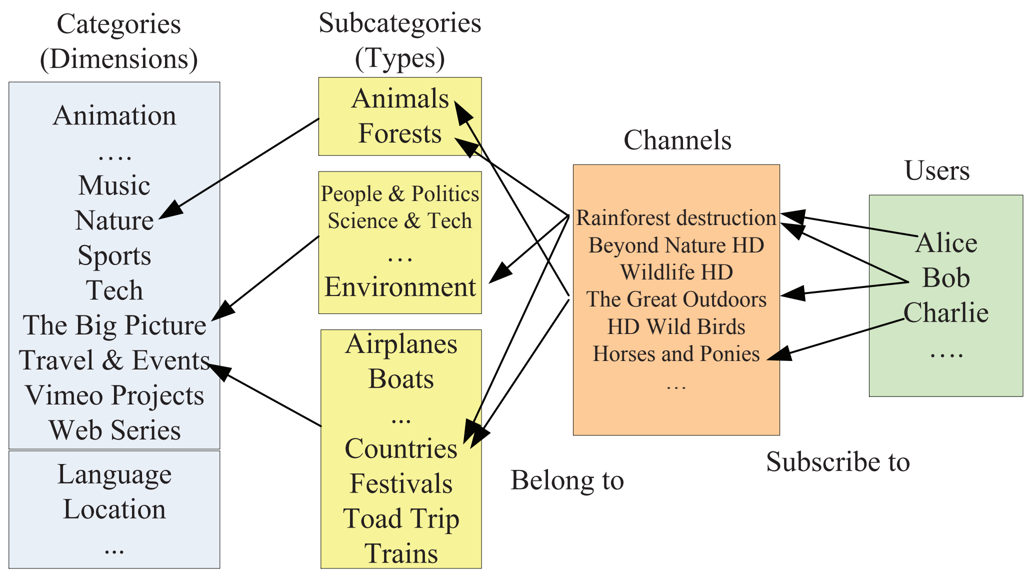


Figure 2.1: Vimeo's feature space structure.

efficiently locating the files. Representative systems for unstructured P2P include PA-VOD [7] and NetTube [8], and the typical structured P2P system is HyperCuP [9]. However, all of these systems are based on a per-video level: for each video, an overlay graph is constructed. As pointed out by paper [1], the per-video overlays not only generate prohibitive costs for maintaining the overlays but also create plenty of redundant links between a pair of nodes on different overlays.

To resolve these problems of the conventional P2P systems, SocialTube [1] leverages users' actual subscription relationships. Many VoD sites provide channel subscription functions to users. A user can create his own media channel web pages, and each channel contains some videos related to its topic. For easily browsing channels, most VoD sites classify channels into categories, according to their features, and each category may have different types of values (i.e. subcategories). Based on the content of video channels, the channel owner and Vimeo staff can associate several category values with the channels. Fig. 2.1 gives an example of the category structure in Vimeo. If a registered user is interested in a channel and wants to track all videos from the channel, he can subscribe to it and keep receiving update feeds whenever new videos are uploaded. Consider that the subscribers of a channel are likely to watch the same videos from the channel and that users with similar interests tend to access similar videos. SocialTube proposes an interest-based per-channel hierarchical overlay: peers subscribed to the same channel are built into one low-level community, and the channels of each category are formed into a high-level community. When a video query is generated, random walk-based searching is conducted within these communities. The per-channel overlay structure can quickly locate the resources when users search for their subscribed channels' videos. Compared to the traditional per-video overlay, the new structure has both low maintenance costs and high QoS.

SocialTube has two problems. First, it is not efficient for subscribers to explore videos belonging to other categories. Users' watching preferences (i.e. interests) are not a fixed value; one may find his new watching preferences by exploring other categories' channels. Take Fig. 2.1 as an example. The subscribers of "The Great Outdoors" channel are also likely to watch the videos from "Rainforest Destruction". By using SocialTube, one has to go through all the searching space of either 'Nature" or "Travel & Events" categories to find the target channel's videos. In addition, if the subscribers of "The Great Outdoors" want to watch videos in the "Animation" category, it is hard to locate the videos on SocialTube's overlay, since there is no efficient overlay connecting across different categories. Second, SocialTube is not applicable for the VoD sites (e.g. Vimeo), who are rich in category information. Note that most channels are naturally suitable to be labeled by multiple categories. For instance, "The Great Outdoors" belongs to both the "Animals" subcategory of the "Nature" category and "Countries" subcategory of "Travel & Events". As a result, the high-level communities, created by SocialTube, are actually overlapped with each other, and eventually form into one community, instead of several communities.

In this chapter, we propose a new hierarchical P2P overlay structure, call SocialVoD, by exploring both the subscription relationships and channels' similarities on Vimeo. The key design of SocialVoD is the utilization of channel subscribers' social closeness: the closer the watching preferences of users are, the better the overlay connectivity should be between them. We not only build interest-based per-channel overlays but also we organize these overlays according to channels' existing category information in the high-level. The main idea of our approach is that, by neighboring the channel overlays, whose category information differs in one dimension, the resulting graph contains enough routing "hints" for seeking other unsubscribed channels' videos. By using SocialVoD, all peers are clustered into a loosely-connected multi-level community structure, which has the small-world property. Based on the new overlay structure, we further propose a routing algorithm for both channel-subscribed users and unsubscribed users. SocialVoD keeps the low maintenance cost feature of previous works, but significantly improves the system's query efficiency, especially when users are developing new watching preferences. Extensive simulation results show the efficiency of our new system in comparison with SocialTube and NetTube.

2.2 Literature Review

The research on P2P architecture has existed for a decade. Many papers have proposed different schemes [10, 11, 12, 13]. PA-VoD [7] and NetTube [8] are the two most classic unstructured-P2P systems. PA-VoD builds a video's overlay by grouping all the peers, who are watching the same video, together. When a user finishes watching, he no longer acts as a provider and eventually will be wiped out from the overlay. PA-VoD works, if there are plenty of users watching the same long video, such as a movie, the typical length of which is 1-2 hours. However, as for the VoD platform, such as Vimeo, most videos are short, the median length of which is about 200 seconds [14]. As a result, a majority of short videos' requests are handled by the central server, instead of peers. The basic idea of the NetTube is that the users, who are watching the same videos, are more likely to view the same content in the future. By associating a cache buffer with each node, NetTube puts the viewers of the same video into one overlay, and further allows them to search for videos from each other. Unlike PA-VoD, the users of NetTube will temporarily store the videos, which were watched previously, in their local buffer. Therefore, after he or she finishes watching a video, as long as the buffer contains the file, the user will remain in the video's overlay. This unique design significantly improves the resource availability in a P2P system. However, both NetTube and PA-VoD create an overlay for each video, which wastes plenty of network resources.

Hypercup [9] is a typical structured-P2P system, which organizes peers into a hypercube structure. Compared to the unstructured system, Hypercup is fault tolerant and suitable for efficient search. However, since Hypercup constructs the overlay by directly connecting peers to peers, as for high churn conditions, where peers frequently join and leave the system, it suffers from huge maintenance traffic. Unlike Hypercup, SocialVoD treats a group of users, who have the same watching preferences, as a node in a hypercube. Although a system may face the frequent connection and disconnection of peers, the absence of a whole group is rare. As a result, SocialVoD has a high searching efficiency and low maintenance costs. The idea of node group-based hypercube has existed for a while [15, 16]. However, the existing schemes neither consider the social property of peers [17] nor are they suitable for short videos; they simply connect all peers of a group to a super-peer and use the super-peers to build a hypercube structure. Since the size of a VoD channel varies greatly from tens of subscribers to millions, and there are plenty of channels having the same category features, the conventional scheme does not suit the modern VoD system any longer. SocialVoD clusters the peers, who subscribe to the same channel, into a community, and further puts similar communities close to each other.

Table 2.1: The distribution of Vimeo channels' associated category numbers

Number of Categories	1	2	3	4	5	...
Percentage (%)	41.94	38.11	8.44	7.54	3.96	0

The design of P2P overlay's structure is highly related with users' data accessing pattern. Unlike traditionally streaming video systems, modern VoD systems, such as Vimeo and YouTube, have social aspects [18]. As shown by papers [19, 20], the date-sharing patterns in VoD systems have a small world property, which suggests that the videos on these systems have strong correlations with each other. As a result, the P2P overlay structures of VoD should also possess the small world property. A theoretical model for small-world networks by paper [21] picture a small world as a loosely connected set of highly connected subgraphs [20]. The structure of our proposed SocialVoD overlay exactly satisfies this structure.

2.3 Vimeo Category and User Interests

Papers [8, 1] show that most videos on VoD systems are short, and there are strong clustering behaviors among users. In this section, we further investigate the category features on Vimeo and the change of users' interests over years.

Vimeo has a total of 23 system-defined categories. By checking the number of channels under each category, we find that there is an obvious gap between a popular group, which has 13 categories, and an unpopular group. We collected the category information of the popular group's channels, each of which has more than 300 subscribers. There is a total of 782 unique channels, and we further compute the distribution of the number of associated categories in Table. 2.1. We find that there are 58.06% channels associated with more than one category feature. Clearly, overlapped categories should be considered during overlay design.

Users may develop new watching preferences over the years. We randomly collect

Table 2.2: The Change of Category Interests Over 5 Years (%)

Year	Narrative	Arts	Music	Video School	Personal	Big Picture
2011	21.7	16.6	41.7	0	3.3	0
2010	41.2	4.8	4.1	4.5	11.4	4.6
2009	35.7	1.8	12.5	0	0	7.1
2008	13.0	6.2	22.5	24.5	4.1	4.1
2007	8.8	9.3	8.1	16.8	7.0	0

Year	Animation	Travel	Tech	Experimental	Comedy	Nature	Sports
2011	10.0	3.3	0	0	0	3.4	0
2010	0	9.4	13.6	0	1.8	4.6	0
2009	14.3	5.4	0	14.3	8.9	0	0
2008	4.1	6.2	0	2.9	4.2	8.2	0
2007	0	12.5	0	7.6	17.9	6.0	6.0

6,000 users' channel subscription times and investigate the change of users' interests (i.e. the category features of subscribed channels), year by year. Table 2.2 shows the changing pattern of a typical user's interests. This user subscribed to 87 channels from 2007 to 2011. We divide the records into 5 groups according to subscription time and count the distribution of the subscribed channels' category features in each year. From the table, we can see that the user becomes more interested in "Narrative" and "Music" related channels, and less interested in the channels from the "Video School" category. Hence, users' watching preferences do change over the years.

Users' watching preferences can be reflected from their channel subscriptions. We summarize the watching pattern of users in the following: (1) Users frequently look for the videos from their subscribed channels; (2) Users are likely to seek other unsubscribed channels' videos, which satisfy their watching habits; (3) The watching habits may change. Users may access certain videos to explore new preferences.

2.4 Overlay Design

2.4.1 Feature Coordinate and Feature Distance

More and more VoD systems, such as Vimeo and YouTube, classify video channels by category. Unlike conventional tags, which are created by individual video owners, category and subcategory are strictly provided by the system. When a user creates a video channel on Vimeo, he should select several appropriate categories for describing it. Later, the Vimeo staff may add more categories or select suitable subcategory values. Once a channel is created, other users may subscribe to it. We adopt u_i to represent a user, c_i to represent a channel. In this chapter, we treat each category as one feature dimension in the category feature space, and we assume that there is a total of m dimensions (i.e. system-defined categories).

The *feature coordinate* of a video channel is defined as $(f_1, f_2, \ldots, f_m)$, where $1 \leq f_i \leq r_i$ and r_i is the number of subcategories in the ith dimension. The physical meaning of value f_i depends on a real system. For example, in the "HD" category dimension, $f_i = 1$ stands for high-definition videos and $f_i = 0$ represents the regular ones, while, in the "Language" category dimension, $f_i = 0, 1, 2, \ldots$ may indicate English, Chinese, Spanish, and et al. Based on the feature coordinates, the category-feature space is partitioned into $\Pi_{i=1}^{m} r_i$ groups, and each channel belongs to one and only one group. We use G_i to represent a hypercube group. Let $\mathcal{F}(\cdot)$ be an operation, which gives the feature coordinate of an entity, and $\mathcal{F}_i(\cdot)$ gives the coordinate value in the ith dimension: $\mathcal{F}(c_i)$ and $\mathcal{F}(G_j)$ shows the feature coordinate of channel c_i and hypercube group G_i, respectively.

The *feature difference set* from a source coordinate $S = (s_1, s_2, \ldots, s_m)$ to a destination coordinate $D = (d_1, d_2, \ldots, d_m)$ is represented by $H(S, D) = \{i | s_i \neq d_i, i \in [1, m]\}$. The *feature distance* is measured as the cardinality of the feature

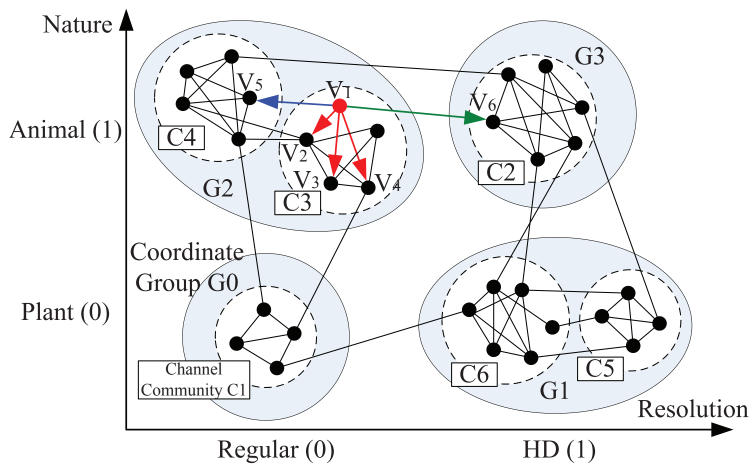


Figure 2.2: A diagram of the network structure of SocialVoD.

difference set $|H|$. For instance, the feature distance set from $(1, 2, 3, 4, 5, 6)$ to $(1, 1, 5, 4, 5, 3)$ is $\{2, 3, 6\}$ since the corresponding values in these three dimensions are different.

2.4.2 Channel Subscription and Peer-to-Peer Graph

In practice, a user may subscribe to multiple video channels, the feature coordinates of which may be different from each other. According to the number of subscribed channels, a user u_i is mapped to one or several nodes $\{v_1^i, v_2^i, \ldots, v_j^i\}$ in the P2P network graph. In this section, we assume that each user subscribes to only one channel, and therefore, u_i is uniquely mapped to v_i. The case that users subscribe to more than one channel will be discussed in the next section.

During the construction of the P2P network, a logical connection is created from users u_i to u_j, at the protocol layer, by letting u_i store the IP address of u_j. The whole system forms a P2P network graph $\mathbb{G} = \langle V, E \rangle$, $E \subseteq V^2$, where $V = \{v_1, v_2, \ldots, v_n\}$ represents the virtual nodes and $E = \{v_i v_j\}$ gives the logic connection from nodes v_i to v_j. Let $g(c_i)$ be a mapping function from a given channel to a set of nodes, who subscribes to c_i; the inverse function $g^{-1}(v_j)$ gives the channels that v_j subscribed to.

2.4.3 Resource Replication Scheme

SocialVoD requires each user to maintain a cache of videos that has been watched within a system-defined session. When the cache is full, the oldest file will be replaced by the new one. This storage scheme can increase the availability of files. Moreover, the popularity of a video changes with time. This caching scheme can eliminate the unpopular files in a timely manner and replace them with the new ones. Here, we are considering a short video, the length of which is usually less than 10 minutes. These files are generally small, and therefore, this cache requirement does not excessively burden users.

2.4.4 Overlay Construction

When a new node v_i appears, it first sends a request to the server, and the server replies with a set of nodes for establishing logic connections. Suppose v_j is a node from the set. According to the features of v_i and v_j, there are three types of connections in SocialVoD:

- If v_i and v_j subscribe to the same channel $g^{-1}(v_i) = g^{-1}(v_j)$, then edge $v_i v_j$ is an *intra-channel* link.
- If they subscribe to different channels but have the same feature coordinate $g^{-1}(v_i) \neq g^{-1}(v_j)$ and $\digamma(v_i) = \digamma(v_j)$, then edge $v_i v_j$ is an *intra-group* link.
- If their feature coordinates differ in one dimension $g^{-1}(v_i) \neq g^{-1}(v_j)$ and $H(\digamma(v_i), \digamma(v_j)) = 1$, then edge $v_i v_j$ is an *inter-group* link.

In order to control the usage of resources, SocialVoD limits the number of out-links that a user can create to b. When the server returns the set, it should guarantee the number of connections for each type following intra-channel : intra-group : inter-group $= p : (1-p)q : (1-p)(1-q)$, where p, q are two system-defined parameters and

$p, q \in (0.5, 1)$.

Take Fig. 2.2 as an example. Assume that v_1 is a new peer in the system, who subscribes to channel c_3, $b = 5, p = q = 0.6$ and $m = 2$. When v_1 contacts the server for the first time, the server randomly selects $b \times p = 3$ peers (i.e. v_2, v_3, v_4), who are online and also subscribe to c_3, to build intra-channel links, as shown by red arrows in Fig. 2.2. The server picks up $b \times (1-p)q \approx 1$ peer from c_4's subscribers, and $b \times (1-p)(1-q) \approx 1$ peer from c_1 and c_3's subscribers to build inter-group links (green arrows) and intra-group links (blue arrows).

The server stochastically creates the connections such that it not necessary to remember the global structure of the overlay graph. When the server seeks the end nodes of the intra-channel links and intra-group links, it randomly picks up the corresponding number of nodes, who are online at the moment. When creating the inter-group links, the server first selects one dimension, in the category-feature space, to be different from it, and then randomly picks up a peer with a different subcategory value in that dimension. This peer is selected as the end of an inter-group link. For a given link, the probability of having two ends with different coordinates is $(1-p)(1-q)/m$. Suppose $\digamma(G_i) = (f_1, \ldots, f_k, \ldots, f_m)$ and $\digamma(G_j) = (f_1, \ldots, f'_k, \ldots, f_m)$. The expected number of out-links from coordinate group G_i to G_j, who differ in the kth category, can be approximated as $b|G_i| \times \frac{|\{v_i | \digamma_k(v_i) = f'_k, i \in [1,n]\}|}{|\{v_j | \digamma_k(v_j) \neq f_k, i \in [1,n]\}|} \times (1-p)(1-q)/m$, where b is the number of out-links per peer, $|G_i|$ gives the number of nodes in hypercube group G_i, and $\frac{|\{v_i | \digamma_k(v_i) = f'_k, i \in [1,n]\}|}{|\{v_j | \digamma_k(v_j) \neq f_k, i \in [1,n]\}|}$ is the probability for selecting f'_k as the new value in dimension k.

SocialVoD creates a two-tier community structure and three levels of overlays. Peers, who subscribed to the same channel, build up the low-tier community, also known as the per-channel community C, as illustrated by C_1 to C_6 in Fig. 2.2. The intra-channel links among the peers build the lowest-level overlay. Because a subscriber is very likely to watch the videos provided by his subscribed channel, clustering peers from the same channel improves files' availability and reduces the average length of query paths. Per-channel communities, who have the same feature coordinate, are clustered into the top-tier community: the coordinate group G, as illustrated by G_0 to G_3 in Fig. 2.2. The middle-level overlay is established by the links connecting different channels within the same group. SocialVoD connects different groups if their feature coordinates are different in one dimension. The inter-group links build up the highest-level overlay. As users tend to access the files with similar features, they may also query the files from unsubscribed channels. SocialVoD puts similar subscribers close to each other, which can improve the query efficiency.

2.4.5 Inter-group Link Management

In SocialVoD, a feature specified query is conducted by forwarding the query messages from group to group via inter-group links. The speed of finding an inter-group link with the specific feature coordinate directly influences the routing efficiency. For accelerating the searching process, the members of each group vote for a leader, who collects the inter-group link information from its members. Usually, the leaders are selected based on their stabilities.

When a node v_i joins a group, he first asks the leader's IP address from his connected peers, who are in the same group. Then, v_i reports all of his inter-group links, together with his own IP, to the leader. When a peer servers as a relay node in an inter-group routing, he should ask the leader for the IP of other members, who have the inter-group links to the next group. The query messages will be forwarded to the next group via these members. Leaders may consume more resources for performing the book-keeping work. We can adopt role rotation and other incentives from the consideration of fairness. When a leader is going to leave SocialVoD, he should find

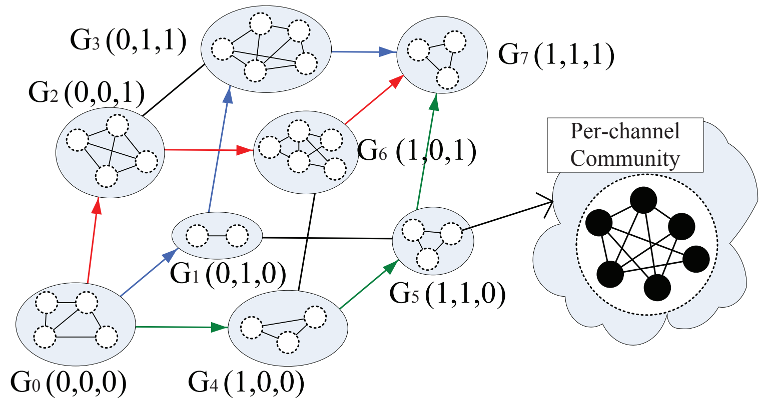


Figure 2.3: Coordinate routing sequence-based inter-group routing.

his inheritor and broadcast this update to group members.

Due to the churn of P2P, peers in SocialVoD periodically check and update their links to other communities. Unlike SocialTube, where peers always update their links to the newest visited peers, SocialVoD removes failed links and replaces them with the links to the other communities, where, recently, the feature-based routing mostly failed. Essentially, SocialVoD gives more emphasis on how to reach a destination, instead of where the destination is.

2.5 Routing in SocialVoD

A query is conducted by two steps in SocialVoD. The first step is called feature-spaced inter-group routing, which uses the highest-level overlay to locate the group with the specific feature coordinate. In order to control the network traffic, each query has a limited number of query copies, assuming h. When selecting query paths, one should avoid the situations where more than one query message searches the same space. The second step of a query is to find the target file within the specific group, and we apply the conventional random walk-based approach to conduct the searching.

2.5.1 Coordinate Routing Sequence and Node-disjoint Paths

In an m-dimensional hypercube, if a source S has a packet for destination D with feature distance k, there are exactly k node-disjoint shortest paths with length k from S to D. Since these paths uniquely partition the searching space into k non-overlapping subspaces, querying along the node-disjoint paths is both efficient and resource-saving. Suppose that S and D differ in k dimensions, where $H(S, D) = \{1, 2, ..., k\}$.

$R_1 : \langle 1, 2, ..., k \rangle$ is defined as the *coordinate routing sequence*, which determines the resolution order of a given set of dimension differences $H(S, D)$ [22]. For instance, the coordinate sequence R_1 requires peers to construct a query path by gradually solving feature differences from dimensions 1 to k. When $H(S, D) = \{1, 2, ..., k\}$, there are a total of k coordinate routing sequences, and the ith sequence R_i : $\langle i + 1, ..., k, 1, ..., i - 1, i \rangle$ is created by making $i - 1$ circular left shifts of R_1:

- The 1st sequence: $R_1 = \langle 1, 2, ..., k - 1, k \rangle$
- The 2nd sequence: $R_2 = \langle 2, 3, ..., k, 1 \rangle$
- ......
- The kth sequence: $R_k = \langle k, 1, ..., k - 2, k - 1 \rangle$

Note that the resolution order in R_1 can be any permutation of the elements in feature difference set $H(S, D)$.

Each sequence uniquely defines one shortest path from S to D. Take Fig. 2.3 as an example. Suppose that G_0 is looking for a video from G_7, and the feature distance set is $H(S, D) = \{1, 2, 3\}$. When S creates $R_3 = \langle 3, 1, 2 \rangle$, the query path according to R_3 is $G_0 \rightarrow G_2 \rightarrow G_6 \rightarrow G_7$, while, if S creates $R_2 = \langle 2, 3, 1 \rangle$, the query path becomes $G_0 \rightarrow G_1 \rightarrow G_3 \rightarrow G_7$. Hence, the coordinate routing sequence set $R = \{R_1, R_2, ..., R_k\}$ creates k node-disjoint shortest query paths from the feature difference set.

2.5.2 The Selection of Coordinate Routing Sequences

In order to control the network traffic, S has only h copies of a query message. In other words, S can conduct h parallel searches for the destination group D. Assuming the feature distance from S to D is k, the relation between h and k influences the selection of query paths. When $h > k$, S is able to use not only the shortest paths for searching D but also the non-shortest ones. However, if $h < k$, S should determine which path (or coordinate routing sequence) has more of a chance to reach the destination group.

Consider that, for a given category, the popularity of subcategories is different. The sizes of some groups are greater than others. Since the ends of inter-group links are randomly and uniformly selected, a larger-sized group absorbs more inter-group links, and has more of a chance to be visited from other groups. As a result, we approximate a coordinate routing sequence's success rate by subcategories' popularity. Suppose the feature coordinates of S and D are $(s_1, \ldots, s_m)$ and $(d_1, \ldots, d_m)$, and dimension $i \in H(S, D)$. In the highest overlay, the transition probability from any group G, who satisfies $\mathcal{F}_i(G) = \mathcal{F}_i(S)$, to other groups G', which have the property $\mathcal{F}_i(G') = \mathcal{F}_i(D)$, is estimated by the following equation:

$$T_i = \frac{|\{v_j|\mathcal{F}_i(v_j) = d_i, v_j \in V\}|}{|\{v_j|\mathcal{F}_i(v_j) \neq s_i, v_j \in V\}|} \tag{2.5-1}$$

where $|\cdot|$ gives the cardinality of a set, and $s_i \neq d_i$. Assume that a path resolves the ith dimension difference first. The successive rate of the path's remaining steps is approximated as follows:

$$T_i' = (|H| - 1)! \prod_{j \in H \setminus \{i\}} T_j \tag{2.5-2}$$

Algorithm 1 The selection of coordinate routing sequences at source node S

$T \leftarrow \phi$
for $\forall G_i \in N(G_s)$ **do**
  for every dimension $j \leftarrow 1 \ldots m$ **do**
    if $\mathcal{F}_j(G_i) \neq \mathcal{F}_j(G_s)$ **then**
      Compute T'_j, and $T \leftarrow T \bigcup \{(j, T'_j)\}$
According to T'_j, sort the elements of T in decreasing order.
Select the first h elements and create routing sequences R.

where H is the abbreviation of the feature difference set $H(S, D)$, and $H \backslash \{i\}$ indicates all the unmatched dimensions between S and D except dimension i.

Algorithm 1 gives the procedure for selecting the coordinate routing sequences when $h < k$. Since the inter-group links are randomly created, a group may not be connected to all other groups, whose feature coordinates differ in one dimension. Assume G_s is the resident group of source S, and let $N(G_s)$ be a set of groups, who are connected with G_s, $N(G_s) = \{G_i | H(\mathcal{F}(G_i), \mathcal{F}(G_s)) = 1, i \in [1, m], \exists v, v' s.t. v \in G_s, v' \in G_i, vv' \in E\}$. For each neighboring group G_i, Algorithm 1 computes the accessing probability T'_j of the remaining feature difference if G_i was selected as the first relay group in the routing path (dimension j is the first resolved feature difference). Algorithm 1 creates the corresponding coordinate routing sequences by selecting the h neighbors, who have the largest T'_j. This method essentially explores the rare inter-group links first. The reason is that, as the processing of the feature-based routing, the searching space becomes smaller; fixing the rare dimensions' values first can increase the success rate of the remaining steps.

Take Fig. 2.4 as an example. Assume that $S = (0, 0, 0)$, $D = (3, 1, 2)$, and $H(S, D) = \{1, 2, 3\}$. The popularity of the 1st dimension "Nature" follows $3 : 6 : 10 : 4$, that of the 2nd dimension "HD" is $4 : 6$, and that of the 3rd dimension "Travel" is $5 : 7 : 3$. There are three potential coordinate routing sequences: $R_1 = \langle 1, 2, 3 \rangle$ (the red path), $R_2 = \langle 2, 3, 1 \rangle$ (the blue path), and $R_3 = \langle 3, 1, 2 \rangle$ (the green path). Using R_1 indicates solving the 1st dimension's difference first.

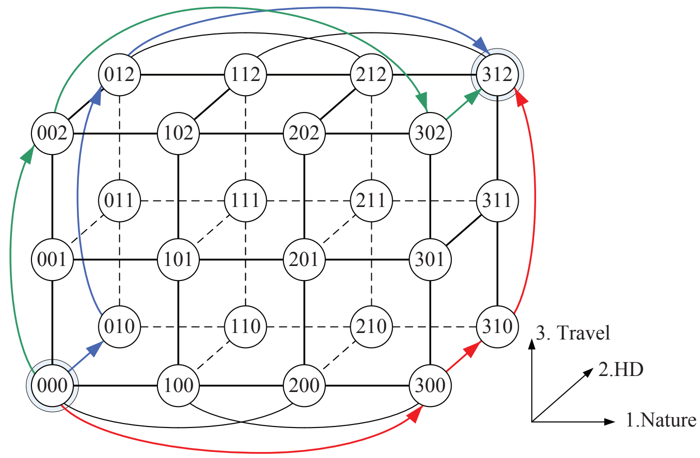


Figure 2.4: The hierarchical community structure in SocialVoD.

For the source node, the remaining success rate of the red path is estimated as $T_1' = (3-1)! \times \frac{6}{6} \times \frac{3}{7+3} = \frac{3}{5}$. Similarly, the rates for using R_2 and R_3 can be calculated as $T_2' = (3-1)! \times \frac{3}{7+3} \times \frac{4}{6+10+4} = \frac{3}{25}$, and $T_3' = (3-1)! \times \frac{4}{6+10+4} \times \frac{6}{6} = \frac{2}{5}$. If $h = 2$, then the source should use R_1 and R_3.

2.5.3 Feature Space Routing

SocialVoD has three types of routings: intra-channel, intra-group, and inter-group. When a user looks for his subscribed channel's videos, an intra-channel routing happens. Query messages are randomly propagated within the per-channel community. Each message is associated with a hop counter, which is decreased by one after each time of forwarding. Once the counter becomes zero, the query message will be discarded. Similarly, if the user seeks for videos from other channels, but have the same feature coordinates, the random walk-based searches are conducted within the corresponding group. However, when the user searches the files from other groups, SocialVoD first sends the messages to the destination group by inter-group routing, and then, adopts intra-group routing to locate the target file within the destination group.

Algorithm 2 Inter-group Routing from Source S

while $h > 0$ **do**
    if $R \neq \phi$ **then**
        /*shortest path routing*/
        Pick up a sequence R_i from R.
        Find $G_j \in N(G_s)$ s.t. $\mathcal{F}_i(G_j) \neq s_i, \mathcal{F}_i(G_j) = d_i$.
        $R \leftarrow R \backslash \{R_i\}$, $mode \leftarrow 0$, send $(R_i, mode)$ to G_j.
    else
        /*non-shortest path routing*/
        Find $G_j \in N(G_s)$ s.t. $\mathcal{F}_i(G_j) \neq s_i, i \in \overline{H}$.
        $\overline{H} \leftarrow \overline{H} \backslash \{i\}$, $mode \leftarrow 1$, send $(H \bigcup \{i\}, mode)$ to G_j
    $h \leftarrow h - 1$

The inter-group routing at a source node is given by Algorithm 2. Based on the relation between h and k, there are two modes, where *mode* is 0 for a shortest path and 1 for a non-shortest path. Assume that the feature coordinate of source S is $(s_1, \ldots, s_m)$ and that of D is $(d_1, \ldots, d_m)$. The feature difference set is represented by $H = \{i | s_i \neq d_i, i \in [1, m]\}$. The coordinate sequences set $R = \{R_1, R_2, ..., R_k\}$ provide the basic guidance for the shortest path mode. Let $\overline{H}$ be the compliment set of H, $\overline{H} = \{i | s_i = d_i, i \in [1, m]\}$, which will be used for the non-shortest path mode.

Algorithm 2 gives the shortest path routing with a high priority; as long as set R contains some coordinate sequences (i.e. $h \leq k$), S should use the corresponding shortest path first. S picks up a sequence R_i from R. Suppose the first resolving dimension of R_i is i. S finds G_j, whose ith dimension value is the same as that of D, from neighboring groups $N(G_s)$. Then, S sends R_i to G_j and asks it to use shortest path mode. When R becomes empty (i.e. $h > k$), it switches to non-shortest path mode, in which the source node increases the feature distance from k to $k + 1$. For instance, in Fig. 2.3, G_0 is sending message to G_3, where $\mathcal{F}(G_0) = (0, 0, 0)$ and $\mathcal{F}(G_3) = (0, 1, 1)$. Although there are only two different dimensions, S can first send the message to group $(1, 0, 0)$ and then forward it to the destination. The path becomes $G_0 \to G_4 \to G_6 \to G_7 \to G_3$.

Algorithm 3 is designed for relay nodes. Assume the current relay node is in group

Algorithm 3 Inter-group Routing for a Relay Node in G_i

```
1: if F(G_i) = F(D) then
2:     Use intra-group routing within G_i.
3: else
4:     if mode = 0 then
5:         R ← R\R[1], the next resolving dimension k ← R[1].
6:         Find G'_i ∈ N(G_i) s.t. F_k(G_i) ≠ d_k, F_k(G'_i) = d_k.
7:         Send (R, mode) to G'_i.
8:     if mode = 1 then
9:         H ← H\{j}.
10:        Find G'_i ∈ N(G_i) s.t. ∃k ∈ H, F_k(G_i) ≠ d_k, F_k(G'_i) = d_k. Send (H, mode) to G'_i.
```

G_i, and the node got the query message from G_0, where $G_i \in N(G_0), \mathcal{F}_j(G_i) \neq \mathcal{F}_j(G_0)$. Let $R[1]$ be the first element in the coordinate routing sequence R. If G_i is the destination group, then intra-group routing will be used to search the file. When G_i is not the destination and the routing is in the shortest path mode, the dimension is different, and has been resolved from G_0 to G_i; it should be removed (step.5). The query message is forwarded to the next group according to the new head of R. As for the non-shortest path mode, any element from the dimension difference set can be used as the next resolving dimension. For example, in Fig. 2.3, G_0 sends $H = \{1, 2, 3\}$ and $mode = 1$ to G_4. At G_4, the unsolved dimension set becomes $H = \{2, 3\}$, and G_4 can forward the message either to G_5 or G_6. One can use Equation. 2.5-2 to find out an optimal receiving group.

2.6 Extension

2.6.1 Multi-channel Subscriptions

The main idea of SocialVoD is to cluster users according to their subscribed channels' category features. However, since a user may subscribe to several channels with different feature coordinates, if we directly use the basic scheme of SocialVoD, peers will form into one giant community, instead of several groups. For instance,

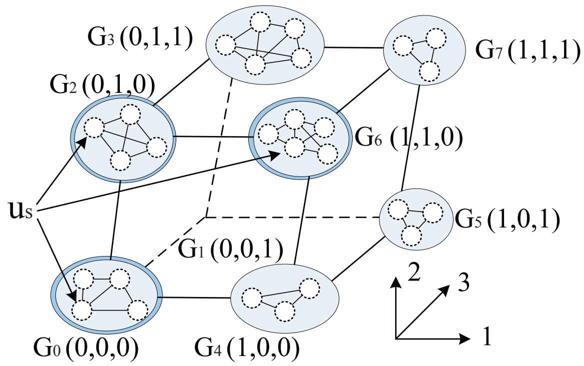


Figure 2.5: Source selection when a user subscribes to multiple channels.

users u_1 and u_2 are the members of channel C, and users u_2 and u_3 subscribe to another channel C'. Directly connecting u_1 u_2 and u_2 and u_3 will eventually put u_1 u_3 into the same community. To avoid this situation, a user is mapped to several nodes on the P2P graph, according to his own subscription. Suppose user u_i subscribed to j channels, then u_i is mapped to a set of nodes $\{v_1^i, v_2^i, \ldots, v_j^i\}$. Each node is associated with b/j out-links (the number of out-links for each user is still b). When searching a file, the user should decide to use which node v_k^i as the source node, since they may have different feature distances to the destination group, as illustrated by Fig. 2.5.

Algorithm 4 gives the procedure for selecting the initial node of a routing. The idea of this scheme comes from Algorithm 1. Let $M(u)$ be the mapping function from user u to his corresponding nodes on the graph, $M(u) = \{v_1, v_2, \ldots, v_j\}$. Basically, Algorithm 4 tests all elements in $M(u)$, and calculates the remaining path's success rate T' from these nodes. Then, it selects the top h paths with the highest success rate. Note that, in practice, not all groups, whose feature coordinates differ in one dimension, are connected. Also, some groups may temporarily not contain any peer (all of them are off-line.) As a result, the nodes, which are closer to the destination in the feature space, may not always be the best source.

Let us go back to the example shown in Fig. 2.4, where $D = (3, 1, 2)$, and the popularity of each dimension follows $3 : 6 : 10 : 4$, $4 : 6$, and $5 : 7 : 3$. Suppose that

Algorithm 4 The selection of initial node for a user u

$T \leftarrow \phi$
for Every $v \in M(u)$ **do**
  Find G_s s.t. $\mathcal{F}(G_s) = \mathcal{F}(v)$.
  for $\forall G_i \in N(G_s)$ **do**
    for $j \leftarrow 1 \ldots m$ **do**
      if $\mathcal{F}_j(G_i) \neq \mathcal{F}_j(G_s)$ **then**
        Compute T'_j, and $T \leftarrow T \bigcup \{(v, j, T'_j)\}$
According to T'_j, sort the elements of T in decreasing order.
Select the first h elements in T, and use them as routing sources.

the user also subscribes to a channel in group $(2, 1, 1)$, then the feature difference set is $H = \{2, 3\}$. There are two potential coordinate routing sequences: $R_1 = \langle 1, 2 \rangle$ and $R_2 = \langle 2, 1 \rangle$. If we solve the 1st dimension first, then we have $T'_1 = \frac{1}{5}$, while, if we solve the 2nd dimension, we get $T'_2 = \frac{7}{10}$. Again, if $h = 2$, then the user should use routing sequence $\langle 1, 2, 3 \rangle$ from group $(0, 0, 0)$ and $\langle 2, 1 \rangle$ from group $(2, 1, 1)$.

When users are allowed to subscribe to multiple channels, the inter-group routing can be accelerated. Here, we propose a new approach, called *feature matching shortcut*. In the classic hypercube routing, each instance of message forwarding can solve only one-dimensional difference. However, in SocialVoD, since a user is mapped to several nodes, it is possible that the physic holder of a query message may be mapped to another peer, who is more than one feature distance closer to the destination. Take Fig. 2.5 as an example. Suppose v_i from G_2 gets a message with G_6 as destination. Since there is another node v_j possessed by the same user u, $v_i, v_j \in M(u), v_j \in G_6, v_i \in G_2$, u can stop the inter-group routing and begin to use intra-group routing from v_j's out-links.

2.6.2 Routing for Non-subscribers

As for users, who have not subscribed to any channel, there is no feature coordinate information. Based on historical query records, SocialVoD deducts the most frequently searched subcategory features of a non-subscriber, in each dimension, and uses these values as the user's feature coordinate. Inter-group links are generated from the user to the corresponding coordinate group, which consists of the subscribers, who have the same feature coordinate. Since non-subscribers' coordinates are not accurate, links are created only from a non-subscriber to subscribers, and not vice versa. When a non-subscriber wants to watch some videos, the inter-group routing will be conducted from the linked coordinate group. Once a nonsubscriber's watching preference is formed (i.e. subscribe to certain channels), most of his links will have already been connected to the subscribed communities.

2.6.3 Pre-fetch for New Videos

One common problem for new videos is their availabilities. Unlike other VoD platforms, Vimeo allows all subscribers of a channel to upload their videos. After a new video is being uploaded, in the initial phase, most queries about it are responded to by the central servers. For improving the availabilities of new videos, SocialVoD uses a pre-fetch scheme, which pushes the new contents to other peers from the same per-channel community. Whenever a new video is generated, the source node sends out several random walkers, carrying the video, to his resided community. When a walker stops, the latest visited peer will store the video in the cache.

2.7 Performance Analysis and Evaluation

In this section, we conduct extensive simulations to evaluate the performances of our proposed system by using both synthetic data and real data. For the ease of comparison, we call our scheme, which clusters the subscribers of the same channel into a community and connects different communities, whose coordinates differ in one dimension, *SocialVoD*; the previous work [1], which only clusters the peers in individual categories, is called *SocialTube*; we also compare the results with

NetTube [8]. Since these approaches use different searching schemes, we keep the maximum possible length of their overall searching paths be the same.

2.7.1 Simulation Setup and Evaluation Metric

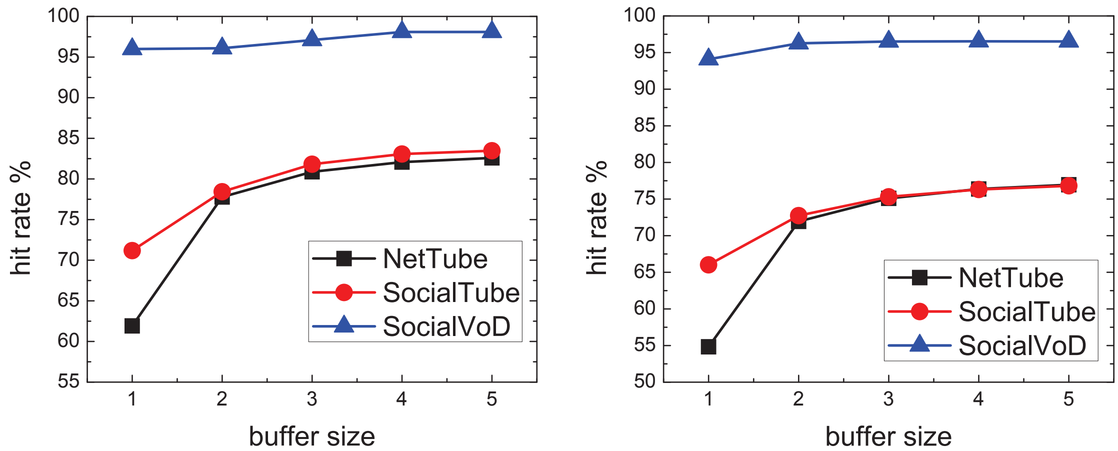
To measure the performance of SocialVoD, we conducted extensive simulations by using Matlab. For the synthetic simulation, we assume that there are 4 categories and each category has 4 different value types (i.e. subcategories); For the real one, we use the real category features of Vimeo channels. We generate $5,000$ peers for the synthetic simulations and $15,000$ peers for the real ones. Each peer is associated with a fixed-size buffer and outlink budget. While setting up, we randomly assign a popularity degree to each subcategory, and each peer randomly selects the subscribed channel coordinates based on the popularity. Therefore, different channels have different sizes. During simulation, some peers randomly create new videos and save into their own buffers.

In this chapter, we emphasize on the following two aspects of users' watching behaviors: 1) frequently watching their subscribed channels' videos; 2) developing new watching preferences by exploring other unsubscribed channels that have similar features. To accurately simulate the watching behaviors, we use the following video selection mechanism. For each dimension, we assign users with a probability for watching the videos with the same subcategory feature. As a result, a user may query the videos from his subscribed channel, and he may also query for other types of videos; users have a better chance of the videos with similar features to their subscribed channels than a totally different video type. After sending a query, a node will wait for a TTL (i.e. time to live). When time is up, the node will send a query directly to the central server and get the video file. No matter whether the node found the corresponding video from the P2P system or not, eventually, it will get the file and locally save it in the buffer.

When building the P2P overlay in the feature space, the number of outlinks from one community to the other is determined by the number of peers subscribed to the channel and system variable $(1-p)(1-q)$. During feature space routing, SocialVoD sends out several query messages with different coordinate sequences. Since the feature space overlay is stochastically established, some messages may not be able to reach the communities with the target's feature coordinates. Once a message reaches the destination communities, it uses a random walk-based scheme to search the videos in the buffers of the community's peers. Later, we test the impacts of message copy numbers on both feature space routing and inner community routing. In our simulation, we are interested in the following two metrics: (1) Average hitting rate: the percentage of queries having found the corresponding videos from P2P overlays. This value shows the percentage of workloads that have been reduced from the server. During simulation, we mainly consider this metric. (2) Average delay: the average waiting time before watching a video. Note that all three approaches first search the videos on P2P overlays, and then resort to the server. Hence, the delay has an upper bound value. Generally speaking, the shorter the average delay is, the better a P2P overlay design is.

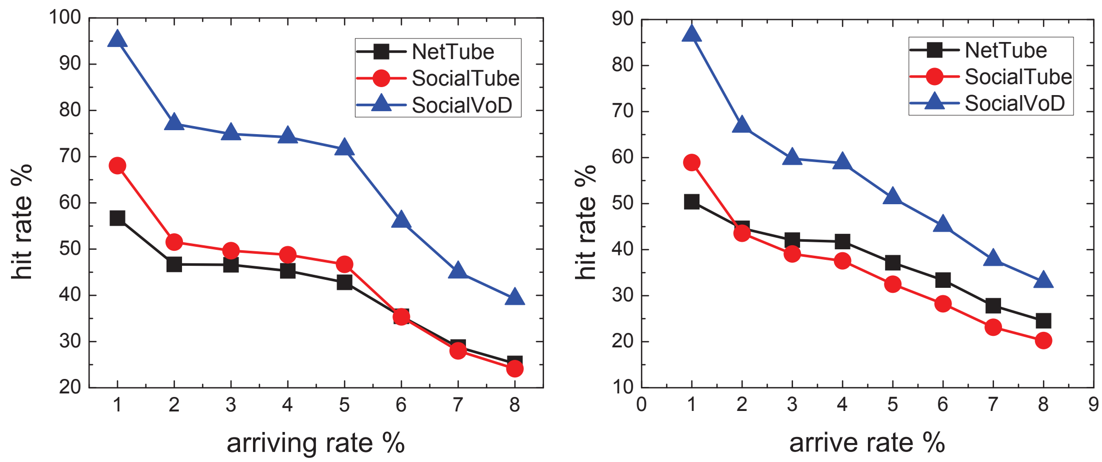
2.7.2 Simulation Results

We first investigate the impacts of the buffer's size. The buffer temporarily stores the videos that have been watched before. Clearly, the bigger the buffer is, the higher the probability is for finding a video by P2P overlay. However, the benefits for using buffers do not grow linearly. In this simulation, we let a peer's query probability be 80%, different dimension's exploring probability be 35%, let 5% of peers generate new contents per simulation session (with there being 20 sessions). As shown by Fig. 2.6,



(a) Uniform distribution (b) Normal distribution

Figure 2.6: The impact of the buffer size.



(a) Uniform distribution (b) Normal distribution

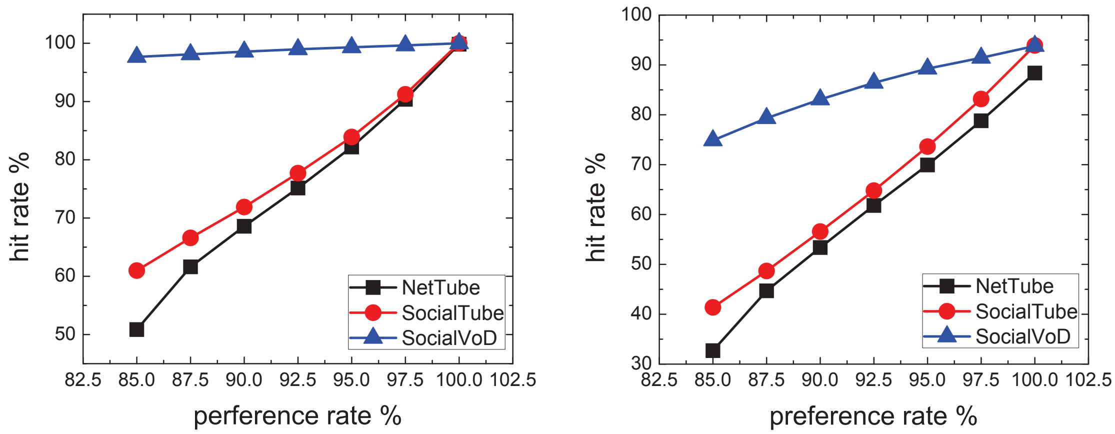
Figure 2.7: The impact of the new content's arriving rate.

with the growth of the buffer, the average hitting rates go up. SocialVoD always has the highest average hitting rate. Moreover, when the popularity of subcategory values follows a normal distribution (the mean value is 25%, the standard deviation is 3.61%, and all values are positive), the hitting rate of SocialVoD is lower than that of the uniform distribution condition. However, SocialVoD still beats others.

In Fig. 2.7, we consider the impacts of the new contents' arriving rates. If a lot of new videos are generated in a short time, only a few peers may cache the files. As a result, with the growth of the new content arrival rate, the average hitting rates quickly drop down, which is much faster than the changing speed of the average delay. In the simulation, we let the generation rate of each peer change from 1% to 8%. We assume that peers know about the new contents. As more contents are generated, users have more choices for the queried contents. If a node's query frequency is slower than the system's new content's generating frequency, the cached files will soon become useless. Here, we let the average query rate of each peer be 75% per session, and the size of a peer's buffer be 1.

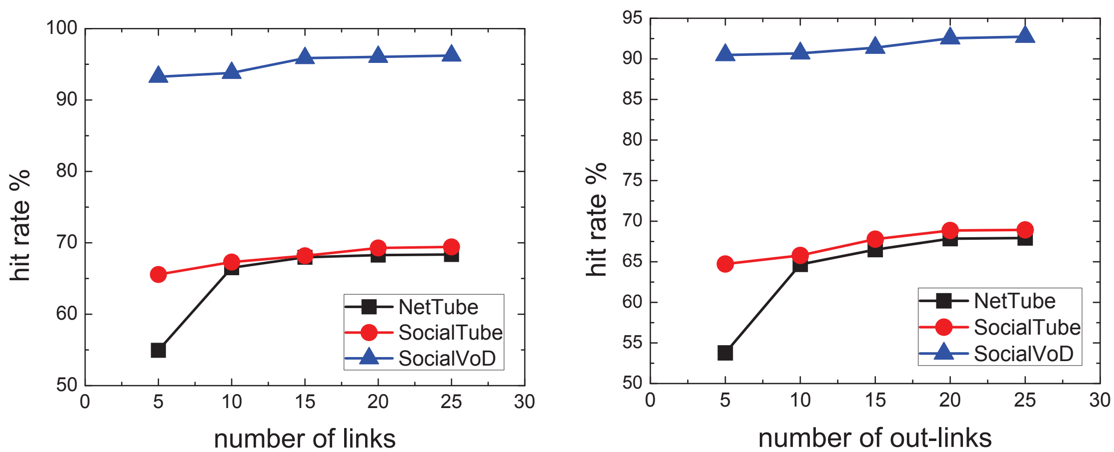
Next, we study the influence of the users' watching preferences. Based on the feature coordinates of users' subscribed channels, each user has his own watching preferences. However, any user may also explore other types of videos with similar feature coordinates. In Fig. 2.8, we gradually decrease the probability of exploring new features for each query from 18.5% per dimension to 0%, and we can see that the average hitting rate of all methods goes up. Note that each video has several dimensions. For example, if the exploring probability is 30% and there are 4 dimensions, the query rate for a peer's own subscribed channel's video is about 24%. Since SocialVoD is specially designed for exploring other video features, its performance is much better than that of others, especially when the system's dimension exploring rate is high.

The number of links that each peer connects to essentially affects the query's efficiency. A well-connected community has a relatively short mixing time, and the random walk-based queries from the same peer can quickly become independent of each other. Moreover, for the overlay in feature space, having more outlinks from a peer can increase the routing success rate. In Fig. 2.9, we gradually increase the number of outlinks per node from 5 to 25, and keep 10% of the outlinks connecting to other communities. We can see that the average hitting rate gradually becomes higher and higher. Note that, due to new content existing, and the stochastic feature of our overlay structure, the average hitting rate cannot reach 100%.



(a) Uniform distribution (b) Normal distribution

Figure 2.8: The impact of users' watching preferences.



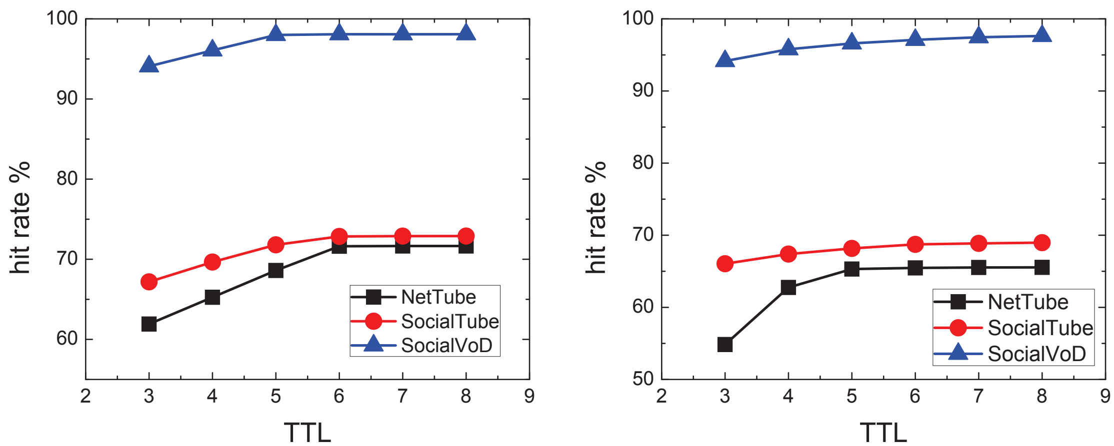
(a) Uniform distribution (b) Normal distribution

Figure 2.9: The impact of each peer's total number of links.

In the next two figures, we study the impacts of the length of the searching paths during the random walk-based query. For SocialVoD, when there are fewer query budgets for inter-community routing, it will only use the shortest paths' coordinate sequences. However, if there are more query paths, then the SocialVoD algorithm can also adopt the coordinates of non-shortest paths. In general, the number of non-shortest paths is much greater than that of the shortest paths. So, the success rate of inter-community will be increased by using more query messages. For all approaches, including SocialVoD, SocialTube and NetTube, the hit rate is also related to the

Table 2.3: The impact of the push operation for newly generated contents

Method	SocialVoD	SocialVoD (push)	SocialTube	SocialTube (push)
Delay	14.369	14.003	26.121	25.912
Hit rate	97.39%	98.35%	77.10%	77.63%

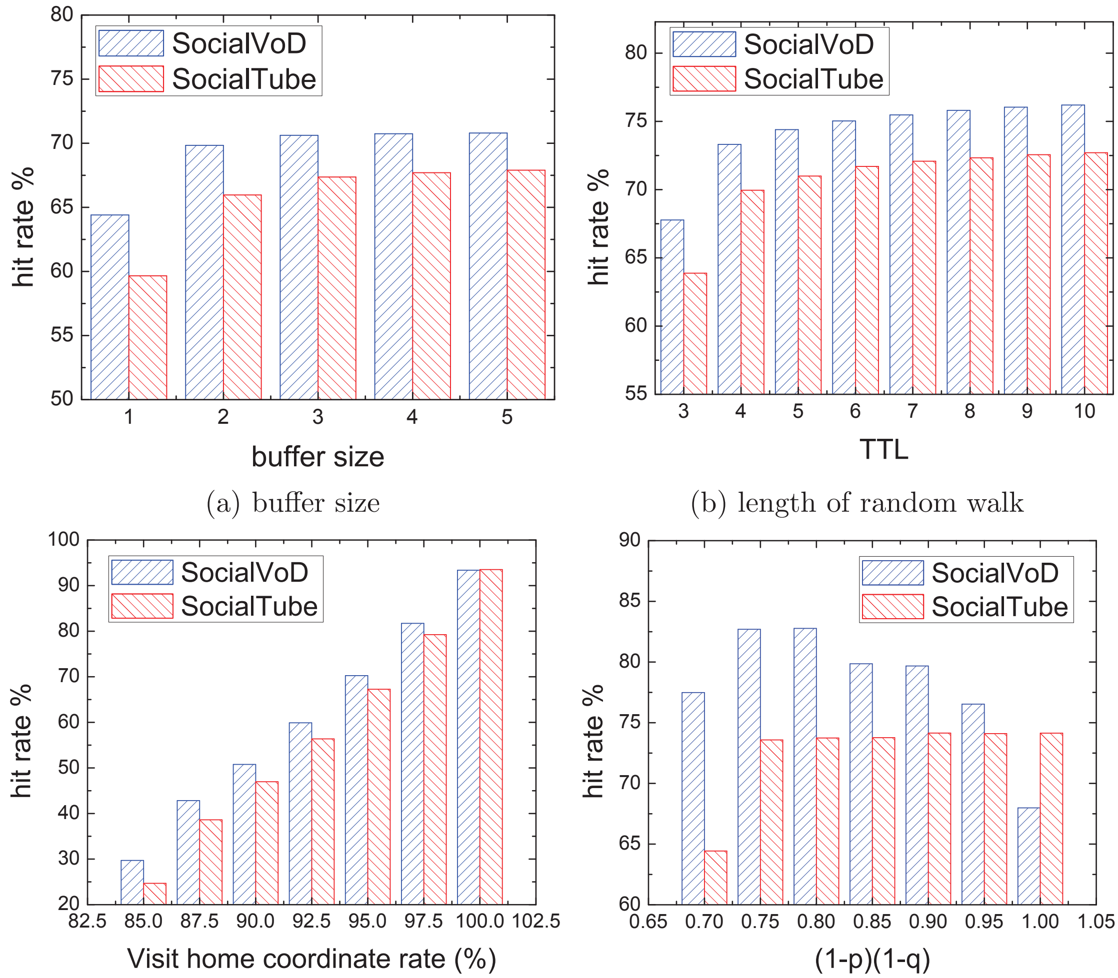


(a) Uniform distribution (b) Normal distribution

Figure 2.10: The impact of the length of random walks.

length of intra-community random walks. In the simulation, we gradually increase the query's TTL of intra-group routing from 3 to 8, and keep these approaches' maximum possible overall length of query paths be the same. Fig. 2.10 gives the results. Because SocialVoD first uses a relatively small number of messages to shrink the searching space of potential peers, it performs better than others in the aspects of hitting rate. Note that, since SocialTube is specially designed for YouTube where each channel can only belong to one and only one category, its intra-category searching becomes less efficient when channels have multi-dimensional features.

In order to increase the average hitting rate, whenever a peer generates a new content, he can push the data to other peers randomly. Essentially, the operation can increase the availability of the new contents, and there are several ways to implement it. For example, from the new content owner, he could send out several copies of the new content and use random walk-based approach to spread the content within his community. Here, we test the simplest case, where the new content is directly pushed



(a) buffer size (b) length of random walk

(c) watching preference (d) community structure

Figure 2.11: Real data-based simulation results.

to the peer's neighbors. Table. 2.3 shows that the pushing scheme can improve the new contents' availability on a P2P overlay. Both SocialVoD and SocialTube have less average delays for the queries of the new contents.

Fig. 2.11 shows the testing results by using real data. We collect the category information of Vimeo channels, who have more than 300 subscribers. In total, there are 782 channels and 13 categories. Users are randomly assigned to different channels, and the average number of users is 12.7. The experiment results of Figs. 2.11 (a), (b), and (c) are consistent with that of the synthetic simulations. In Fig. 2.11 (d), we test the impacts of the value of $(1-p)(1-q)$, which controls the percentage of outlinks connecting to other communities that have different feature coordinates. There is a trade-off between the number of inter-community links and intra-community links. With the growth of intra-group (or inter-group) links, searching files from the resident community (or other communities) becomes easier. From Fig. 2.11 (d), we find that both approaches are affected by p and q; However, SocialVoD gets more influence. The reason for this phenomena is that, when looking for videos with different features, SocialVoD utilizes the inter-group links for routing, while, SocialTube independently seeks files from each related dimensions. As a result, SocialVoD gets more impacts when there is few inter-group links (i.e. $(1-p)(1-q) = 1$). But, for most cases, SocialVoD still has better hit rate than SocialTube.

2.8 Chapter Summary

Over the past decade, we have witnessed an explosive growth of online video-on-demand (VoD) services, such as Vimeo and YouTube. With the growth of the services' popularity, the scalability, bandwidth, and delay problems of the conventional client-server architecture have become increasingly obvious, which has compelled the development of the peer-to-peer architecture.

In this chapter, we explore the existing social relationships (i.e. common channel subscription) and social similarity (i.e. the similarity of users' watching preferences) to establish a new P2P overlay structure. Unlike other VoD systems, the users on Vimeo have more social interactions: any subscriber of a channel is allowed to upload his videos to the channel, and all subscribers can watch and discuss the videos' contents within the channel. Clearly, in such a high-interaction platform, a channel's subscribers are likely to watch the videos from the same channel. Moreover, users' watching preferences gradually vary, and therefore, they also tend to watch other unsubscribed channels' videos, which have similar content features (i.e. category values). Based on these properties, our system, SocialVoD, creates a hierarchical overlay among peers: subscribers of the same channel form into a low-level community, and in the high-level overlay, different channels' groups are connected based on their similarity. This unique structure has three significant features. First, the SocialVoD provides relatively short query paths when users are searching for the videos, the features of which are close to their subscribed channels. The second advantage is that SocialVoD can efficiently locate the low-level communities with specific features via the high-level overlay structure. Third, compared to the conventional per-channel overlay structure, in SocialVoD, each peer has low maintenance costs.

Here are the main contributions of this chapter:

1. We find that the existing P2P systems do not fully utilize the social properties behind the users' watching patterns. We introduce a new hierarchical P2P overlay structure, which has the small-world social feature. Considering the users' watching patterns also present a small-world phenomenon [20], our new structure is more suitable.

2. Our new system explores the existing category information as an overlay's construction "guide" and routing "hints", which significantly reduces the searching space of videos. We design a new routing algorithm, which creates several node-disjoint searching paths within the P2P overlay. The algorithm can be applied to both channel subscribers and non-subscribers.

3. We provide extensive simulations to show that our new system can efficiently locate the files and is applicable on a large scale.

CHAPTER 3

EXPLORING SOCIAL MOBILITY IN THE DESIGN OF MOBILE CLOUDS

Social information has various types. Chapter 2 explores the subscription behavior of users and their common interests during the construction of P2P overlay. In this chapter, we consider another type of social information, *social contact's pattern*, and incorporate it into the decision-making process of workload allocations in a mobile cloud application.

There are plenty of idle computational resources on the Internet, which could potentially be used for accomplishing huge tasks. More and more applications are being designed for exploring those idle resources. In this chapter, we focus on the idle computational resources, including both human intelligence and machine computing abilities, in mobile social networks (MSNs). Based on the unique features of MSN, we design a new cooperative system, called social-crowdsourcing. The distributed and infrastructure-free features of the system make it more attractive than traditional crowdsourcing platforms. In the proposed system, a huge work is gradually partitioned into smaller pieces and is propagated from node to node. However, how to partition and allocate these segments is a critical problem, which directly affects the work's completion time and system throughput. Due to the lack of global information, independent relay nodes are likely to make conflicted decisions, which will cause an unbalanced workload distribution on participating nodes. In this chapter, we find that, for a work at different processing stages, one should adopt distinct workload exchanging schemes, moving from a progressive method to a conservative one. Based on this observation, we propose an adaptive workload allocation scheme, in which a participating node can gradually switch his decision strategy according to the workload statuses of neighboring nodes. By using our approach, system throughput can be significantly improved, and large works can finish within a nearly optimal time. Unlike in traditional scheduling problems, we take a human's rejection, contact delay, and social similarity into consideration. Extensive simulation results show that our proposed algorithms can successfully make full use of the idle resources in MSNs.

3.1 Introduction

Around us, plenty of computing resources have been wasted, including both human intelligence and machine computational abilities. For electronic devices, such as smartphones or tablet computers, we may leave them turned on for a whole day without doing anything to them; even when we are using them, only a portion of the machine's computational resources have been utilized, let alone the human intelligence that is wasted each day. In order to make better use of these idle resources, several centralized crowdsourcing projects have been implemented, such as Boinc [23], Folding@home [24], and Amazon MTurk [25]. In these projects, a project owner uploads a large and time-consuming task onto a server in advance [26], and volunteers participate in certain parts of the task when they are idle. The tasks may relate to human intelligence, such as seeking an object from images, or the tasks may simply use the computational resources of idle machines, such as training a model or solving complex mathematic functions.

However, there are two constraints, which restrict the development of crowdsourcing systems. First, the existing crowdsourcing platforms, such as Amazon MTurk, lack an advertising mechanism to timely recruit participants (also known as workers): It is hard for a newly created task to attract enough participants in a relatively short time unless the task owner gives a very attractive payment. Many off-line workers, who are eager to do certain types of tasks, are not able to be timely aware the existence of the new tasks. Moreover, plenty of people do not even know of the existence of certain platforms, let alone the tasks on them. Secondly, the current system is centralized and platform-specified, and therefore, it is not flexible and robust enough. The workers, accustomed to one platform, are less likely to participate in another platform's tasks, and the unavailability of these platforms completely destroys most of the existing crowdsourcing applications.

In this chapter, we create a distributed and self-organized crowdsourcing scheme within mobile social networks (MSNs), called *social-crowdsourcing*. The main idea of the social-crowdsourcing scheme is that, after recruiting a worker, the task not only gets the workers' abilities, but also the resources potentially contributed by the worker's related people. Our scheme creates a multilayered outsourcing structure and explores the idle computing resources within *social domains*. Instead of waiting for others to log into a crowdsourcing platform, select your tasks, and work on them, our system directly sends the task to the potential workers via multi-hop social contacts. By using the scheme, users can self-organized build up a crowdsourcing system for a task, or it can also be used as an extension of workers in conventional crowdsourcing systems.

Example task in social-crowdsourcing: One may meet the following situation: when thinking about a problem, we recall that we have read a paper mentioning an idea, which could be used. But we do not remember which article it is. Although we have saved all of the papers in electronic documents, finding the article is still hard since that idea may be semantically described. Solely going through all of the documents' contents is extremely time-consuming; instead, we can recruit friends for seeking the article.

The procedure of social-crowdsourcing: A task owner (a mobile device's user) first creates a social-crowdsourcing task by including both the electronic documents and job description. Once it is done, the owner becomes the first worker and begins to locally search the article. Note that the task owner could be the owner of the documents, or a worker, who undertakes a portion of a task from the conventional crowdsourcing platform. In social-crowdsourcing, any participant can further recruit new workers via any kind of social contacts. For example, when a worker physically comes across his social contactor, such as friends or colleague, the participant's device automatically sends a message to the contactor and ask his willingness to participate in the task. If the answer is yes, a portion of the documents will be transferred from the worker to the contactor via shortwave radio, and then, both of them can go through the papers in parallel. When a participant is able to access free Internet, such as in vicinity of a free WiFi access point, or the participant agrees to use cellular networks to transmit social-crowdsourcing's workload, he can directly send the helping message or transfer a certain amount of workloads to his friends via network-storages, such as email, instant messaging (IM), or Dropbox. Later, when his friends get Internet connection (e.g. 3G/4G or free WiFi), their devices automatically fetch the data. When a participant finds the result, he returns the article's id to the task owner via the cellular network. Note that only a few participants need to return and the size of the package is very small.

Remuneration: For workers' payment, we adopt MIT DARPA Challenge Team's scheme: the winner who finds the article gets half of the total bonus; the worker, who invited the winner, gets one-fourth of the total; the people who invited the inviter gets one-eighth, and so on. The main advantage of this incentive scheme is that users, who do not want to personally seek the paper from the documents, are still willing to participate and propagate the task to their social contacts.

During work segments' dissemination processes, the estimation of assigned

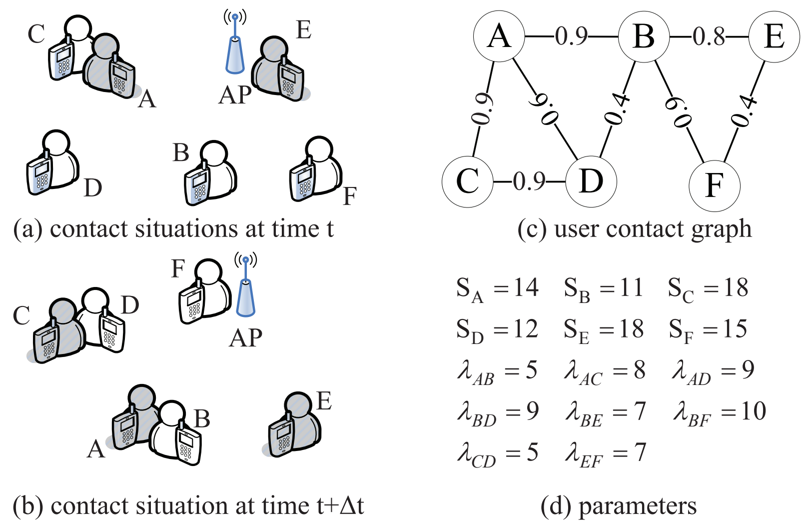


Figure 3.1: Workload allocation in social-crowdsourcing.

workloads to each social contact is a critical problem. Essentially, a selected allocation algorithm directly affects not only the whole work's completion time but also the resource utilization rate of the social-crowdsourcing system. In this chapter, we address the following question: given huge works originating at random nodes, by what strategy can the works' segments be appropriately disseminated during stochastic contacts, such that the system's throughput can be maximized? Due to the distributed feature of our system, this problem is not trivial. Take Fig. 3.1 as an example, where shadowed users indicate the ones carrying work segments and idle users are represented by the color white. When A meets C, A should leave some extra workload to C, such that C can forward these extra work segments to his future encounter D at an earlier time. When A transmits work segments to B, he must also count B's potential workloads given by B's other friends. In short, a worker has to take others' future potential contacts, accepting probabilities, contacting delays, current workloads, processing speed, and the impacts of common friends into account.

To solve the problem, we first propose a distributed workload allocation algorithm based on the historical information of worker's 1-hop neighbors. Consider the fact that, no matter who owns the work and how large the work is, a worker's overall computing ability should be a fixed value; we further propose a distributed algorithm for pre-estimating each node's future computing ability. By comparing the work completing progress patterns of these two algorithms with those of the optimal ideal case, we finally propose an adaptive solution. The adaptive scheme automatically switches the work allocation methods according to the work segments' propagation status. Our solutions not only work under the single work, single source condition, but also are applicable in the situations of multiple works and multiple sources. Extensive simulation results show that our proposed schemes can significantly increase the system's overall throughput.

3.2 Literature Review

In crowdsourcing [27, 25, 28], a tedious work is often subdivided into smaller pieces, which are then assigned to an undefined group of workers [29]. However, the current crowdsourcing systems are centralized [30]. Workers proactively join a crowdsourcing platform and seek tasks. When using a crowdsourcing platform [31, 32], inevitably, someone has to pay a fee for using the centralized server. For example, currently, Mturk collects a 10% commission on top of the amount that you paid for your work [33]. Moreover, these centralized platforms cannot assign works to the workers if there is not an internet connection. Paper [34] proposed an archetype for building a distributed crowdsourcing system in DTN. In this chapter, we extend their model on mobile social networks, and further analyze the schemes for opportunistically allocating work segments among participants. When determining nodes' workloads, our schemes take contacting delay, computing speed, acceptance probability, the number of initial work sources, and the existence of resource competition works, into consideration.

Scheduling is a process of determining how to commit machines (executants) among a variety of works. Based on the number of executants, scheduling can be categorized into single-machine scheduling [35] or multiple machine scheduling [36]. Generally speaking, the scheduling method used in this chapter belongs to the multiple machine scheduling. As the name suggests, multiple machine scheduling tries to make up schedules for different executants. However, in our model, data segments are transferred based on nodes' opportunistic contacts. We cannot explicitly assign an exact work segment or workload, to a specific machine in advance, like the traditional multiple-machine scheduling did.

Scheduling in distributed systems involves load-balance and congestion control [37]. For the heterogeneous system, where nodes have different computing abilities, partitionable workloads are considered. Usually, a task is divided into different-sized segments, according to the capability of their executants [38, 39]. But in our problem, the size of an assigned workload is not only related to the local computing ability of a node, but also the abilities of the node's future contacts. Backpressure routing [40] is an algorithm for dynamically adjusting traffic over a multi-hop network. However, backpressure routing is hard to implement in a fully-distributed style, and it aims at enhancing the throughput of a network, while, our problem focuses on designing a local algorithm for reducing the completion time of works. Adopting backpressure routing may not be a good solution for our problem.

3.3 System Model and Problem Formulation

3.3.1 System Model

We consider a typical MSN, which consists of M mobile users (also named nodes). Let V_i represent a node. Each node is associated with a constant processing speed S_i, and we assume that, once a node participates in a task, it will not stop until the task has been finished. Social-crowdsourcing involves multiple rounds of recruiting via social contacts: a user's workload will be outsourced to his friends, friends' friends, and so on. Essentially, a huge work is gradually partitioned and propagated from friend to friend. In our model, a work consists of several work segments, and a work segment is the smallest uncleavable data unit. The workload of a node is the total number of work segments that have been assigned to it. Any user could be a task's owner (also called a source node), and more than one task could exist in the system.

In social-crowdsourcing, information exchanges during any kind of social contacts: locally or remotely. For local contacts, two nodes should physically encounter each other (like nodes' contacts in DTN), and data is transmitted via shortwave radio; remote contacts indicate the situations that two remote friends communicate with one another via cellular network or free WiFi-based Internet. In practice, data transmission of remote contacts can be implemented by using network storages, such as email, IM, or dropbox. When a worker contacts his friend for the first time after participating in a task, the friend can determine whether to join in the task. We use P_{ij} to represent the accepting probability. Once agreed, their devices will automatically adjust the workloads whenever they contact other participants of the task until it is completed. For any worker, he may physically encounter other participants, stochastically. For the participants, who agree to use cellular networks at any time, they may query the working progress of their friends and adjust the workloads, once in a while. As for the workers, who use free Internet to transmit workload, they also can do same things whenever they get free Internet. Clearly, the contacting interval between a pair of workers is a random variable, and we use λ to represent the average inter-contacting time. Table 3.1 summarizes the common symbols used in this paper.

Note that we cannot let all friends of a worker share one network storage "box" due to the security and trust reasons. Social-crowdsourcing has a concept of commitment: at any time, a work segment only belongs to one worker. If we put the workload in

Table 3.1: Common notations

Notation	Description
G	nodes' social contact graph, $G =< V, E >$
V_i	a participant (user/node)
M	total number of nodes
S_i	user V_i's local computing speed
$S_i^{(p)}$	user V_i's future potential ability
P_{ij}	the accepting probability between V_i and V_j
$Q_{ij}(t)$	encounter (contact) probability between V_i, V_j within t
λ_{ij}	average inter-meeting time between V_i, V_j
$N(\cdot)$	the neighbor set of V_i on contact graph
$E(\cdot)$	the expected value
W	wasted computing resources
ΔW	exchanged workload
Φ_i	the assigned workload on node V_i
A	workload transition matrix $A = [A_{ij}]$
L	the size of the original work

one box and let the worker's friends to fetch the data based on their dynamic needs, some workers may maliciously modify others' work segments in order to prevent them from winning the bonus (the friends of a user may not be friends). In order to avoid using complex security mechanisms, we do not let multiple users share a network storage.

3.3.2 Problem Formulation and Challenges

Social-crowdsourcing is a distributed system. From a source node, how to allocate the segments of a given work during stochastically contacts becomes a fundamental problem, which directly influences the completion time of the work. *The goal of the chapter is to opportunistically maximize the overall throughput of a social-crowdsourcing system.* Since work segments are transmitted hop-by-hop, at each instance of segment relay, a relay node must estimate how many workloads he will keep, and how many workloads should be forwarded to the social contacts. The optimal allocation happens when (1) each participating node begins to work as early as possible and (2) all participants complete their assigned work at the same time. However, due to the lack of global real-time information and the accurate future encountering times, it is impossible for nodes to provide an optimal allocation.

The social-crowdsourcing system can be abstracted as a stochastic bucket network: each node is a huge bucket, and there is a hole at the bottom of each bucket. The size of the holes may be different (which means the computing speed of each node may be different). Stochastically, we can pour some amount of water from one bucket to another (which models the workload reassignments during social contacts). Initially, only one bucket has water (the original large job), and we want to find a local scheme to determine the amount of transmitted water such that the water in the whole system can be drained as quickly as possible. For the rest of the chapter, we will not discriminate how a social contact is conducted (locally or remotely) since all types of social contacts can be abstracted as stochastic data exchange between friends.

In order to estimate the size of transmitted workloads during social contacts, the relay nodes must be able to answer the following two questions: 'how to describe a node's local computing ability and the overall abilities of his future encountered nodes?' and 'how to estimate the carrying workloads of the future potential contacts?' Since segments are disseminated based on multi-hops relay, the participants must carry an appropriate workload for their future contacts.

The estimation of future encountered nodes' capacities is difficult, not only because of the uncertainty of their contact times and acceptance decisions, but also the double counting of the abilities of their common friends. For instance, Fig. 3.1 (c) is a possible contacting graph among six users. When V_A forwards workloads to V_B, he should not only consider V_B's capacity, but also the potential workloads are given by V_D, V_E, and V_F; otherwise, V_B may overload. Also, when V_A forwards workloads to V_C, he should make a decision on whether to give V_D's work segments to V_C for the further relay. Obviously, V_A could also keep the segments and directly forward them to V_D later, or give them to V_B. Even if V_A selected a relay path to V_D with the highest probability, the realistic shortest contact path could be another one. Note that our problem is essentially different from the conventional delay-tolerant networks (DTN) studies. In our problem, any idle node could be a work segment's consumer. Since the optimal result happens when all workers simultaneously complete their workloads, we should avoid the unbalanced workload distribution among workers, which has never been considered in DTN.

3.4 Solution Overview

3.4.1 Key Observations

Compared with traditional online social networks and delay-tolerant networks, MSNs have several unique features, which influence the performance of a solution. In this part, we present some important features of MSNs on our problem.

The number of participants, in the initial phase, highly influences the whole work's completion time. Roughly speaking, the number of participants at this phase affects the speed of work segment propagation. This observation is consistent with the research results in the field of maximizing social influence. In our model, work segments are disseminated to participating nodes via multiple relays. If a node does not carry enough work segments, he may locally process all the segments before contacting other idle nodes. Progressively estimating the abilities of a new participant and its future contacts could reduce the likelihood of this condition happening, but could also cause the imbalanced distribution of work segments.

In social-crowdsourcing, it is inevitable to have multiple segment-relaying flows pass the same node. However, whether the confluence is beneficial or not is determined by whether the situation could potentially reduce the work's completing time. During the initial phase of segment propagation, flow convergence may result in overloads, while, at a later time, it also provides chances for re-balancing the workload.

Due to the lack of global real-time information, it is extremely hard to estimate the future potential encounter nodes' carrying workloads. However, the size of transferred workloads essentially relates to the workloads' differences between a pair of contacts, instead of their absolute workloads.

3.4.2 Main Idea

In this chapter, we propose an adaptive scheme for allocating work segments. Instead of assigning the accurate workload to a node on the first try, we *fuzzily* divide the whole process into three phases. Phase one intends to assign an approximately corrected amount of workload onto each region; phases two is used for dynamically re-balancing the workloads within small regions, and the final phase ends the work by opportunistically transferring the unbalanced workloads to idle nodes.

In the first phase, since a majority of nodes have not obtained the work yet, we use *an aggressive scheme to estimate nodes' overall computing abilities*, including the potential abilities from the future contacts. Therefore, for each node, we consider a *local computing speed* and a *future potential speed*, which indicates how much help a node could potentially obtain from future opportunistic contacts. Since there are time delays and rejecting probabilities at each time of transmission, computing the exact value of the potential speed is inappropriate and impossible. Considering that the aggressive scheme cannot provide a fine adjustment of workloads, instead, we take the overall speed that a node could get via the opportunistic contacting paths as the criteria. This metric can integrate transmission delay and accepting rate together, and avoids the double-counting problems caused by common friends. Although the work segments are physically propagated via random paths, our metric essentially computes the expected sum of speed that the whole system could provide to the node.

As for the second phase, when two users contact each other, they use their stored neighbor's historical workload status information to estimate the work's finishing time, and further make workloads' adjustments based on it. The second phase is the transition phase. The intuition behind this mechanism is that, once each node's neighbors have the same completion time, the workloads are balanced and distributed among all participating nodes. By exploring the two contacts' direct neighborhood information, one extends his local view to 2-hop. Based on an estimated completion time distribution, a node proportionally assigns weights to the aggressive scheme and the conservative scheme.

The locally stored historical information used in phase two is collected during a pairwise contact. It may have already become outdated before making a workload allocation decision; some nodes could complete their assignments earlier than others. At the final phase, nodes simply transfer some unfinished work segments to the randomly encountered contacts. The method, used in this phase, purely considers each node's 1-hop neighborhood status and that of the current contacts.

3.5 Solution Details

3.5.1 Contact Frequency and Contact Graph

We assume that the inter-contacting time t between any two nodes V_i and V_j follows exponential distribution with the pairwise contact rate λ_{ij} ($\lambda_{ij} > 0$). The contacting probability density functions (PDF) are represented as follows:

$$Q_{ij}(t) = \lambda_{ij} \times e^{-\lambda_{ij} \cdot t} \tag{3.5-1}$$

where $Q_{ij}(t)$ stands for the contacting probability between nodes V_i and V_j within time interval t. Assume that there is a r-hop transmitting path from node V_i to node V_r with edge weights $\lambda_{i,1}, \lambda_{1,2}, \cdots, \lambda_{r-1,r}$. According to [41], the total transmitting time follows hypoexponential distribution:

$$Q_{ir}(t_1 + t_2 + \cdots + t_r) = \sum_{s=1}^{r} C_{s,r} \times \lambda_{s,s+1} e^{-\lambda_{s,s+1} \cdot t} \tag{3.5-2}$$

where $C_{s,r} = \prod_{u \neq s, s=1}^{r} \lambda_{u,u+1} / (\lambda_{u,u+1} - \lambda_{s,s+1})$ and $t = t_1 + t_2 + \cdots + t_r$.

After iteratively exchanging information with friends, nodes learn the system parameters within r-hop, such as computing speed, average accepting rate, and average inter-contacting time. Based on this information, a contact graph is locally created on each node, which records the identities of a node's contacts, together with the system parameters. Besides the system parameters, each node may also store certain auxiliary information about its direct contacts, such as a node's potential ability (which is discussed in Chapter 3.5.4). Note that, in our system, a node may also record others' expected completion times. However, such values are not in real-time, and the node can only learn distant nodes' expected completion times via multi-hop contacts. For instance, in Fig. 3.1 (c), node V_A never contacts V_F, but it can learn V_F's expected completion time via node V_B's historical estimation. After encountering with V_F, V_B may estimate its finishing time based on the carrying workload at the contacting time, and record the value together with a time stamp. When V_A meets V_B, he could update his record about V_F by using the newest historical estimation.

3.5.2 Neighbor Status-based Finishing Time Estimation

In our system, after each instance of pairwise contacts, a node is able to estimate the work's finishing time within its 1-hop neighborhood. The estimation of finishing time is not trivial, due to the fact that nodes may adjust their workloads among other neighbors at any time. Therefore, the recorded carrying workload of a neighbor node may have already been changed at the time of estimation.

Algorithm 5 shows our procedure for approximating the finishing time. Without loss of generality, we assume that Algorithm 5 is running on node V_u, and node V_v is the current contactor of V_u. Since a user V_i could be the common contactor of both nodes V_u and V_v, the first step of Algorithm 5 is to eliminate the double counting problem. We virtually split the common contactor V_i into two virtual nodes and use its accepting probabilities to weight each virtual node, as shown by Algorithm 5, line 5. Essentially, line 5 splits the carrying workload and the local computing speed according to the corresponding accepting probability, and exclusively virtually assigns a portion of the carrying workloads and speed to V_u and V_v, respectively. The advantage of this approach is that, after virtually splitting, the completion times of the two virtual nodes are still equal to those of V_i. Via this way, node V_u is able to rationally estimate the real possible impacts of his neighbors' workloads and computing abilities.

Although workload adjustments may happen at any time, here, *we are only interested in a set of time periods, which could potentially come at the cost of wasting computing resources*. More specifically, within node u's 1-hop neighborhood, computing resources are wasted if workload adjustment happens during the condition that some neighbors have finished all of their workloads while others are still working. After getting an approximated value of the wasted computing resources, u can estimate his finishing time by first summing it up with the real workloads, and then

Algorithm 5 Neighbor Status-based Finishing Time Estimation (NSFT)

1: /*Assume NSFT is running on u and the other encounter is v*/
2: Find u and v's common neighbors $N(u) \cap N(v)$
3: **for** $\forall i \in N(u)\backslash v$ **do**
4:     **if** $i \in N(u) \cap N(v)$ **then**
5:         Split the recorded workload Φ and speed S of the common neighbor of u and v: $\Phi_i \leftarrow \Phi_i \times \frac{P_{iu}}{P_{iu}+P_{iv}}$, $S_i \leftarrow S_i \times \frac{P_{iu}}{P_{iu}+P_{iv}}$
        /*Eliminate the double counting problem of the common neighbors of u and v*/
6: Based on u's local record, estimate each neighbor's finishing time $\{\Phi_i/S_i\}$, and Sort $\{\Phi_i/S_i\} \cup \Phi_u/S_u$ in ascending order
7: **for** $\forall i \in N(u)\backslash v$ **do**
8:     **if** $\Phi_i/S_i < \Phi_u/S_u$ **then**
9:         Compute the expected wasted computing resource W according to Equation (3)
10:         /*Wasted resource exists when u is working while i is idle*/
11:     **if** $\Phi_i/S_i > \Phi_u/S_u$ **then**
12:         Compute the wasted resource W_u by Equation (3), and update a temporary variable tp: $tp \leftarrow \min\{tp, W_u/S_u\}$
        /*Resource is wasted when u is idle while the neighbors of u are working*/
13: $W \leftarrow W + (\sum_{\forall i, i \in N(u)\backslash v, \Phi_i/S_i < \Phi_u/S_u} S_i + S_u) \times tp$
14: Estimate the finish time t_F:
$t_F \leftarrow [\sum_{\forall i \in N(u)\backslash v} \Phi_i + \Phi_u + W] / [\sum_{\forall i \in N(u)\backslash v} S_i + S_u]$

dividing the result by the summation of weighted speeds, as shown by Algorithm 5, line 14.

From the viewpoint of V_u, he only needs to care about the workload adjustment between him and his neighbors. For the neighbor node $\{V_i\}$, who will finish its current workload before V_u (Algorithm 5 line 8), we can estimate the wasted computing resource at a node V_i, W_i, as the product of V_i's computing speed and the expected time delay after V_i finishes the work. Assume t_{next} is the next contacting time between nodes V_u and V_i, then we have:

$$\begin{aligned} W_i &= S_i \times E[(t_{next} - \Phi_i/S_i), \Phi_i/S_i < t_{next} < \Phi_u/S_u] \\ &= S_i \times \int_{\Phi_i/S_i}^{\Phi_u/S_u} (t - \Phi_i/S_i)\, \lambda e^{-\lambda t}\, dt \end{aligned} \tag{3.5-3}$$

For each node V_i, who completes the corresponding works before V_u, we need to calculate a W_i and sum them up (Algorithm 5 line 9). However, as for any node V_j,

Algorithm 6 Local 1-hop Status-based Workload Adjustment

/*Assume it is running on u and the other encounter is v*/
Get v's 1-hop information from v
$\Delta W \leftarrow |\Phi_u - \Phi_v|/2$
while $NSFT(u, \Delta W) \neq NSFT(v, \Delta W)$ **do**
  if $NSFT(u, \Delta W) > NSFT(v, \Delta W)$ **then**
    if $\Phi_u > \Delta W$ **then**
      $\Phi_u \leftarrow \Phi_u - \Delta W$, $\Phi_v \leftarrow \Phi_v + \Delta W$
    else
      $\Delta W \leftarrow \Phi_u$
  else
    if $\Phi_v > \Delta W$ **then**
      $\Phi_u \leftarrow \Phi_u + \Delta W$, $\Phi_v \leftarrow \Phi_v - \Delta W$
    else
      $\Delta W \leftarrow \Phi_v$
  $\Delta W \leftarrow \Delta W/2$
Send/receive ΔW amount of workload to the other contactor

who finishes his works after V_u (Algorithm 5 line 11), only V_u's wasted computing resources are considered (since V_u records only 1-hop information, the wasted resources during the workload propagation directly between V_u's 1-hop neighbors are ignored.) Therefore, for the node group with a completion time larger than V_u, we only need to find the earliest completing time: after this time, V_u may get new work segments again (Algorithm 5, line 12.) One may notice that, when V_u finishes his workloads and is in idle status, the neighbors with earlier completing times must also be idle. So, we also estimate their wasted resources during the period from V_u completing his workload to getting new segments (Algorithm 5, line 13).

3.5.3 Local 1-hop Status-based Workload Allocation

Whenever two users contact each other, they could reallocate their workloads such that the work can be completed at the same time. Intuitively, after exchanging and updating their recorded neighborhood information, each node can figure out an ideal size of the transferred workloads by using NSFT algorithm. Here, we adopt the binary search algorithm to estimate the size. Note that, in our model, both contacts could

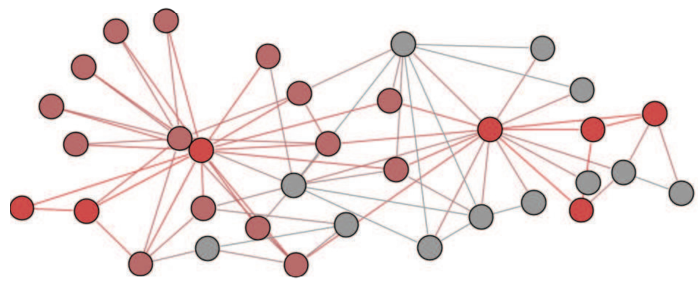


Figure 3.2: Contacting graph of example data set.

run the NSFT algorithm and get the same value of the transferred workload, ΔW. Algorithm 6 gives the procedure.

In order to check the effectiveness of this 1-hop information-based algorithm, we compare it with the optimal result and the naive result, respectively. In the naive case, after two nodes meet each other, workloads are reallocated according to their local processing speed. The optimal result is based on posterior knowledge: after events have happened, all nodes' contacting times and accepting decisions are recorded and known. Based on this information, the optimal scheme simply finds out the fastest relay paths from the work owner node to any other participants. The optimal scheme calculates the earliest work segment arriving time of each node, and then, assigns an appropriate amount of workloads to the participants, such that they will finish the job at the same time.

Figs. 3.3 to 3.7 illustrate the workloads' average diffusion patterns on a regular contacting graph (as shown by Fig. 3.2) with the growth of observation time. The contacting graph is based on karate club social graph [42]. When generating these figures, we keep each node's acceptance decision fixed, and repeat the simulation 10 times by randomly creating different encountering times. On these graphs, the intensity of the gray color at position (x, y) represents the carrying workload on node y at time x. We use the absolute work amount to represent the unfinished workloads being carried on each node. The darker a line's color is the more workloads that are

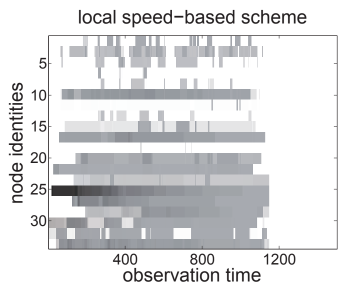


Figure 3.3: Single work: case 1.

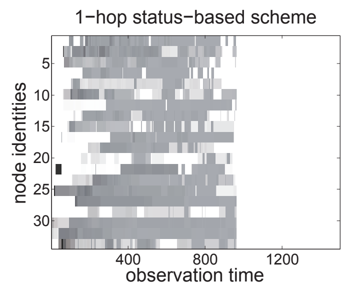


Figure 3.4: Single work: case 2.

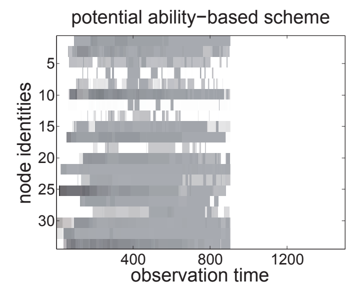


Figure 3.5: Single work: case 3.

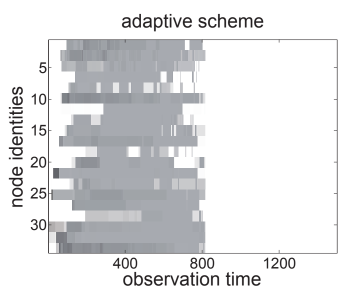


Figure 3.6: Single work: case 4.

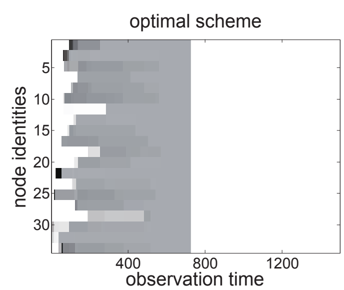


Figure 3.7: Single work: case 5.

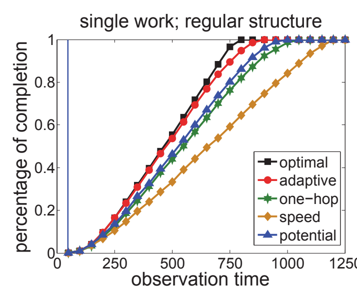


Figure 3.8: Work progress comparison.

held by a node. If a node becomes idle, we use white color to represent it. The work's processing progression is shown in Fig. 3.8. In order to more accurately describe the contributions of different schemes, we also consider an extreme social contacting graph, where the nodes form a chain and each node only has contact with the previous two nodes and next two nodes. The simulation results of this special contacting graph are given by Figs. 3.13 to 3.12.

From Fig. 3.3, we can see that, under the naive scheme, the work segments do not fully spread. Many nodes barely have work segments, and for some other nodes (such as nodes 1 to 5), they have not obtained appropriate amount of workload: there are many white-colored intervals between the gray one. From the x-axis, we can see that it takes almost $1,200$ units of time to finish the work. In Fig. 3.8, the slope of the brown diamond curve indicates the overall processing speed by using the

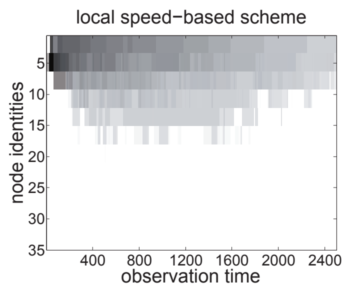


Figure 3.9: Special structure: case 1.

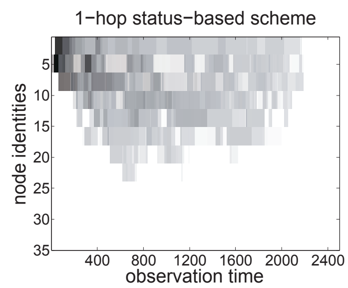


Figure 3.10: Special structure: case 2.

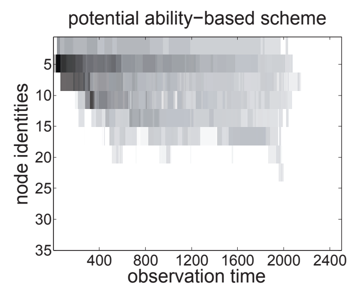


Figure 3.11: Special structure: case 3.

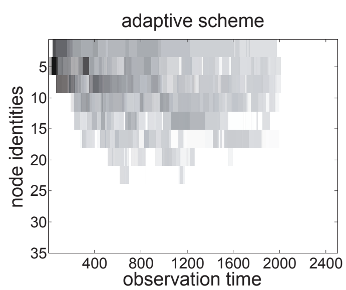


Figure 3.12: Special structure: case 4.

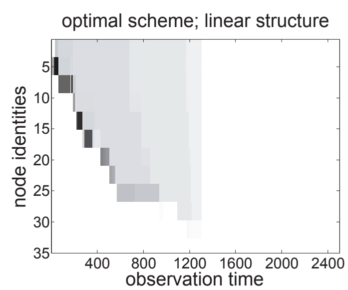


Figure 3.13: Special structure: case 5.

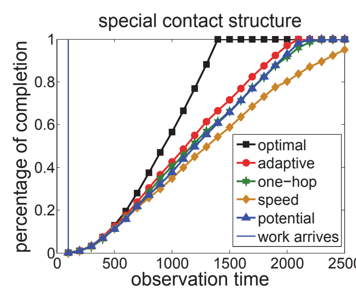


Figure 3.14: Work progress comparison.

naive scheme. In this example, since nodes fail to transfer an appropriate amount of the workload to other nodes, only a limited amount of computing resources is used. Moreover, during the simulation, we also find that the completion time of this scheme is unpredictable, and is influenced by the contacting sequences. As for the optimal result, shown by Fig. 3.7, we can see that, during the initial phase (the first 200 units of observation time), work segments are quickly spread among nodes. As we have mentioned, in the optimal result, all participants complete their work at the same time.

Compared with the naive scheme, the 1-hop status information-based algorithm (Fig. 3.4) can quickly disseminate the workloads during the initial phase; however, from a long-term perspective, the transferred amount at this phase is inappropriate, and therefore, some nodes are out of work quickly. By further checking the size of assigned workloads, we find that the scheme underestimates certain nodes' potentially idle abilities, especially during the initial phase. From the view of the percentage of completed work, as shown in Fig. 3.7, we observe a better result in which, the slope of the green-star curve is larger than the brown diamond one. The simulation result on the chain-style contacting graph gives us a similar result. By comparing Figs. 3.9 and 3.10, we can clearly see that the 1-hop status information-based algorithm can better propagate work segments than the naive scheme. However, by observing the locations of the darkest points between Figs. 3.10 and 3.13, it seems the initial participants should transfer more work segments to the contacts.

3.5.4 Potential Ability-based Workload Allocation

Influenced by the HITS algorithm [43, 44], which is used for rating Web pages, we propose an algorithm for measuring and ranking a node's potential future computing ability that the node could obtain by recruiting other nodes. Essentially, we let each node be associated with two features: a local computing speed and a potential speed. However, unlike the traditional link analysis problems, the links in our model are associated with both accepting rate and random contacting delay. For addressing this problem, we design a special weight on each link, which can successfully integrate the number of friends, contacting delays, and accepting decisions together.

3.5.4.1 Weights of Contacting Edges

For a node V_u, we consider the following situation: if all neighbors of V_u are idle and only V_u has a huge work with size L, what is the expected amount of workload that each of V_u's neighbors, V_v, can obtain? We use the NSFT algorithm, which is discussed in Chapter 3.5.2. However, the NSFT assumes that a pair of nodes has already met. In the current problem, nodes V_u and V_v may or may not come across each other. Similar to previous analysis, we need to measure the expected amount of wasted computing resources, $E[W_v]$, before V_u meets V_v and transfers workload to it. Here, we estimate the wasted workloads as $E[W_v] = S_v \times E[\text{inter-contacting time}] = S_v/\lambda_{uv}$. Let Φ_u and Φ_v be the workloads on nodes V_u and V_v after encounter. They should satisfy the following two conditions:

$$L = \Phi_u + \Phi_v + S_v/\lambda_{uv} \tag{3.5-4}$$

$$NSFT(\Phi_u) = NSFT(\Phi_u) \tag{3.5-5}$$

We use ρ_{uv} to represent the expected percentage of workloads that node V_v can get from V_u.

$$\rho_{uv} = P_{uv} \times \frac{\Phi_v}{L} \tag{3.5-6}$$

where P_{uv} is the accepting probability between V_u and V_v. Without loss of generality, we define ρ_{uu} as the following:

$$\rho_{uu} = \frac{E[L - \Phi_v]}{L} = \frac{\sum_{v \in N(u)} [P_{uv} \times (L - \Phi_v)]}{L \times \sum_{v \in N(u)} P_{uv}} \tag{3.5-7}$$

By normalizing ρ_{uv} for all of u's neighbors, we get a workload transition matrix $A = [A_{uv}]$.

$$A_{uv} = \frac{\rho_{uv}}{\sum_{v \in N(u)} \rho_{uv}} \tag{3.5-8}$$

For each node V_u, the computation of A_{uv} for each neighbor node V_v only requires node V_u's 1-hop information, and therefore, each node can locally compute the transition weight A_{uv} to each neighbor. Note that the transition matrix A is not a symmetric matrix, and $\sum_{v \in N(u) \bigcap V_u} A_{uv} = 1$.

Theorem 3.1. *The transition matrix, $A = [A_{uv}]_{M \times M}$, has a stationary distribution,*

$\pi_{1\times M}$*, such that* $\lim_{k\to\infty} A^k = \boldsymbol{1}\pi$*, where* ***1*** *is the column vector with all entries equal* 1.

Proof. We can model the propagation process of a single work segment by a time-homogeneous Markov chain. Each user (node) is a state. The state transition probability is given by the transition matrix A, which is introduced in Chapter 3.5.4. According to Theorem 4.1 in book [41], a Markov chain has a unique stationary distribution if it is irreducible and aperiodic.

According to our scheme (Algorithm 7 lines 3 to 6) for generating the transition matrix A, all considered states are accessible by the algorithm executing node V_u. Since the communication between nodes is bidirectional, any state in A can access any other state by following a transition path through node V_u. Therefore, the Markov chain is irreducible.

In our model, the average inter-contacting time between any pair of nodes is greater than zero. Therefore, the value of A_{uu} is non-zero. A state in a Markov chain is aperiodic if the state's transition probability to itself is greater than zero. Since all states in our model have the self-loop, the Markov chain is aperiodic. Based on the above two features of the transition matrix A, the corresponding Markov chain has a stationary distribution $\pi = [\pi_1 \quad \pi_2 \cdots \pi_M]$. $\pi_j = \sum_{i=0}^{\infty} \pi_i \times A_{ij}$ $(j \geq 0)$, and $\sum_{j=0} \pi_j = 1$. $\square$

3.5.4.2 Estimation of Potential Ability

During the computation of node V_u's potential ability, both the local computing ability of a neighbor node V_v and its potential ability should be regarded as V_u's potential ability. However, on a social contacting graph, since each node has more than one contactor, a node must equally split its abilities among the neighbors. Here, we adopt A_{uv} as the weights for splitting. Let $S_u^{(p)}$ represent V_u's potential ability, then it can be recursively defined as following:

$$S_u^{(p)} = \sum_{v \in N(u)} A_{vu} \times \left[S_v^{(p)} + S_v \right] \tag{3.5-9}$$

where S_v is the local computing speed of a node, and A_{vu} is the transition probability from node V_v to V_u. One may note that the value of A_{uv} is related to the selected value of L. In reality, instead of computing a single value, we calculate a series of different values of $\langle L \rangle = \langle L_1, L_2, \cdots, L_k \rangle$, and weigh the corresponding results.

There are two common ways to compute the potential ability. The first approach is similar to the computing procedure of the HITS algorithm: each node may simply store a temporary score $S_u^{(p)}(t)$, iteratively exchange the recorded score with neighbors, and update the score by using $S_u^{(p)}(t+1) = \sum_{v \in N(u)} A_{vu} \times \left[S_v^{(p)}(t) + S_v \right]$. Ideally, after several rounds of exchange, $S_u^{(p)}(t)$ becomes stable. However, at the end of each iteration, the approach must normalize each node's recorded score by the total of the current scores; otherwise, the scores will not converge.

The second approach is based on the random walk. By analyzing the definition function of $S_u^{(p)}$, we can see that the value of $S_u^{(p)}$ is essentially the sum of the computing speeds that each node can contribute to V_u along the opportunistic contacting paths. In the second approach, each node sends out several random walkers. Each walker is associated with a score, which indicates the amount of speed that the originated node can contribute to the walkers' current resident node. Whenever a walker passes a node, the resident node records the current score of the walker, and the walker randomly selects its next destination from the direct neighbors of the resident node. Note that, with each transition from node to node, the walk's carrying score will be reduced according to the edge weight, A_{uv}. A walker stops moving when its score becomes less than a pre-defined threshold. The potential ability is approximated by the sum of a node's recorded scores. Although the second approach is fully distributed, in reality, the whole process may take a long time and involve a lot of data packages.

For simplifying the problem, we let each node gradually learn other nodes' A_{uv} and speeds via several rounds of node pairwise contacting. Each node locally creates the whole social contacting graph and computes the potential ability. One can compute the value of $S_u^{(p)}$ either by directly using equation 3.5-9, or by first calculating matrix A's stationary distribution π and then computing $\mathbf{S} \times \mathbf{1} \times \pi$, where $\mathbf{S}_{1\times M}$ is the speed vector.

3.5.4.3 Workload Allocation

Once obtaining the potential abilities of direct contacts, a pair of encountering nodes can redistribute their workloads according to their potential abilities. Algorithm 7 shows the procedure for potential ability-based work segments allocation. The basic idea of Algorithm 7 is that, during the propagation of work segments, especially at the initial phase, if participating nodes can roughly estimate the overall ability of each work assigning flow, the work segments could be disseminated more uniformly. Note that the potential ability is a rough estimation about a participant's future computing capacity. In practice, one should consider both the potential ability and local speed by assigning certain weights to them (Algorithm 7 line 13).

Fig. 3.5 shows the diffusion process after using Algorithm 7. This approach has a better performance than does Algorithm 6, and it assigns a more appropriate amount of work segments to the participants. From Fig. 3.5, we can see that a large part of participants (nodes 15 to 34) are assigned with more workloads within the first 400 units of observation time. The completion progression of a given work is presented in Fig. 3.8; From the very beginning, Algorithm 7 beats Algorithm 6. However, based on the algorithms, we know that the 1-hop status-based approach's local workload

Algorithm 7 Potential ability-based Workload Allocation

/*Assume the algorithm is running on V_u*/
/*Generation of the edge weights in matrix A*/
for Every node $v \in N(u)$ **do**
  for Every L_i in $\langle L_1, L_2, \cdots, L_K \rangle$ **do**
    Compute $A_{uv}^{(L_i)}$ for each sampling workload size
Compute the weighted average of A_{uv} based on the size distribution of real works
/*Compute V_u and its neighbors' potential ability $S^{(p)}$*/
Learn other nodes' A_{uv} and S_v via pairwise contacting
Compute potential ability by equation 3.5-9
/*Workload allocation scheme on node u*/
Exchange and update state information with v
if V_u is contacting V_v and $\Phi_u + \Phi_v > 0$ **then**
  $\Phi \leftarrow \Phi_u + \Phi_v$, $\alpha \leftarrow (S_u + \beta S_u^{(p)})/(S_u + S_v + \beta S_u^{(p)} + \beta S_v^{(p)})$
  $\Phi_u \leftarrow \alpha\Phi$, $\Phi_v \leftarrow (1-\alpha)\Phi$

adjustment is better than that of the potential ability-based scheme: by further comparing Fig. 3.4 with Fig. 3.5, we can see that 1-hop status-based approach use more idle resources on nodes 5 to 15. However, in Figs. 3.10 and 3.11, we did not see such a difference. Probably due to the chain structure, a 1-hop status-based approach cannot equilibrate the workloads well. Based on the respective advantages of the 1-hop status-based scheme and the potential ability-based scheme, we wonder whether we could combine them together and get a better result.

3.5.5 Adaptive Scheme

Workload allocation process can be fuzzily partitioned into three phases. Initially, most nodes are idle, but their realistic accepting decision is unknown. In this phase, the large-scale propagation of workloads, in a balanced way, should be the first priority of an allocation scheme. As more and more nodes participate in the cooperative work, the amount of exchanged workloads gradually becomes smaller, and the undertaken workload of each node trends to stable. Therefore, at the initial phase, we allocate the workloads mainly based on each node's expected potential ability. Here, we adopt the potential ability-based workload allocation method (Algorithm 7). Because most nodes have not participated in the new work, and the workload partition decisions are made based on the average condition, the workload allocation scheme used in this phase is more progressive than those of the later phases.

As time grows, the accepting decisions of a majority of nodes become clear. The workload allocation process gradually enters the second phase, which focuses on balanced distributing workloads among the real participating nodes. A typical feature of phase two is that all participating nodes are physically working on the same work's segments, simultaneously. Nodes in our model essentially perform two operations: processing work segments one-by-one, and re-balancing the remaining workloads among the contacts within several hops (these nodes virtually build up a region). Although a certain amount of workload has been transmitted from the source node to each region, due to the uncertainty of nodes' accepting decision, the realistic workload distribution among different regions may not be balanced. Therefore, we combine the potential ability-based method with the local 1-hop status-based approach (Algorithm 6): the former approach helps to re-balance workloads between different regions, and the latter one re-allocates the workloads within each local region.

During the final stage, some nodes complete their work and become idle again, while some other nodes are still working. Opportunistically and locally re-balancing has the top priority. Consider the fact that the local 1-hop status-based approach holds more accurate neighborhood completion time information than the aggressive one. Hence, at the final stage, we focus on using the 1-hop status-based scheme.

Our adaptive solution combines the potential ability-based approach together with the local 1-hop information-based one. However, how to identify the three phases discussed above, and how to smoothly switch between the two approaches, is critical. Clearly, at the initial stage, the expected work completion times are

Algorithm 8 Time Adaptive-based Scheme

/*Assume the algorithm is running on node V_u*/
for Every contact between V_u and its neighbors V_v **do**
    Exchange and update state information
    Compute the completion time's distribution $\{p_i\}$
    $\alpha \leftarrow 1 + \log^{-1}(M) \times \sum_{\forall i} p_i \log(p_i)$
    Compute ΔW^p and ΔW^l by calling Algorithms 6 and 7
    $\Delta W \leftarrow \alpha \times \Delta W^p + (1-\alpha) \times \Delta W^l$

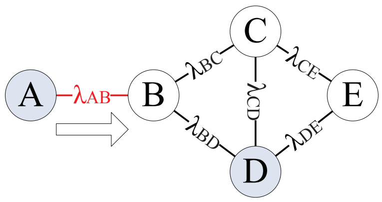
highly unbalanced, where the expected finishing times for the initial participants are much higher than others. Meanwhile, at the final stage, the completion times are more uniformly distributed. Since there is not a clear boundary between each stage, we use the distribution of the expected completion times as an indicator, which is able to quantitatively describe how much closer the current system status is to the initial stage (a highly biased distribution) and the final stage (a uniform distribution). However, since the global real-time statuses are not available, the completion times' distribution is purely based on each node's local recording of the finishing time information about other nodes. As we have mentioned in Chapter 3.5.1, a node may learn other nodes' expected completion times by multiple rounds of information exchanging between different nodes.

We adopt the entropy of the completion times' distribution [45] as the metric. Let ΔW^p be the amount of transferred workloads by using the potential ability-based scheme, ΔW^l be the transferred workload by adopting a local 1-hop information-based scheme, and p_i be the percentage of nodes with completion time t_i. The transferred work amount in our adaptive scheme is determined as follows:

$$\Delta W = \alpha \times \Delta W^p + (1-\alpha) \times \Delta W^l \tag{3.5-10}$$

$$\alpha = 1 + \frac{\sum_{\forall i} p_i \times \log(p_i)}{\log(M)} \tag{3.5-11}$$

where α is a mixing parameter, and M is the total number of a work's participants.



(a) user contact graph

$S_A = 1$	$\lambda_{AB} = 1/5$	$\lambda_{DE} = 1/5$
$S_B = 0.01$	$\lambda_{BC} = 1/5$	$W_A = 1000$
$S_C = 1$	$\lambda_{BD} = 1/10$	$W_D = 1000$
$S_D = 10$	$\lambda_{CD} = 1/5$	$W_B, W_C, W_E = 0$
$S_E = 30$	$\lambda_{CE} = 1/10$	

(b) parameters

Figure 3.15: An example about workload allocation in social-crowdsourcing.

When $p_i = 0$, we assign zero to the computing result of $p_i \times \log(p_i)$. During the computing of α, we compare the entropy of the current completion times' distribution with that of the uniform distribution, which is equal to $\sum_M 1/M \times \log(1/M) = -\log(M)$.

Algorithm 8 shows the procedure of our adaptive scheme. Note that more than one work may coexist in our system. Since the goal of this chapter is to maximize the system's overall throughput, here, we do not discriminate the segments from different works. Figs. 3.6 and 3.12 show the workload diffusion pattern by using the adaptive scheme. By comparing Fig. 3.6 with Figs. 3.4 and 3.5, we can clearly see that the adaptive scheme takes the advantage of both the potential ability-based scheme and the local 1-hop status-based scheme: in the initial stage, workloads are widely and appropriately disseminated among participants, and at the later stage, workloads are locally equilibrated among neighbors. In Fig. 3.8, the proposed adaptive solution is the best approximation algorithm. However, as for the simulation result on the special chain-style contacting graph, although the adaptive scheme still beats all other solutions, its excellence is smaller than that of the simulation on the regular contacting graph.

3.5.6 Example

Take Fig. 3.15 as an example, where shadowed users indicate the ones carrying work segments and idle users are represented by the color white. Assume that, at time t, nodes V_A and V_B come across each other for the first time (after the work arrived the system), and Fig. 3.15 (b) gives the parameters about participants. The carrying workloads of V_A and V_D are 1000, respectively. All other nodes have zero workloads. After the contact between V_A and V_B, intuitively, V_A should transfer a large portion of workload to V_B, since V_B has more potential computing resources. However, if we assign workload only based on local speed, V_B can only get $0.01 \times 1000/(1+0.01) \approx 10$ units of workload. According to the current 1-hop information of V_B, the 1-hop status-based scheme first considers the available computing resources within V_B's neighborhood except V_A, and then it assigns 834 workloads to V_B such that the expected finishing times of V_A and V_B are 166 and 167. Note that the estimation of the finishing time also considers the impacts from the neighbors of V_B. If we adopt the potential ability-based scheme, the overall potential abilities of V_A and V_B are 1.87 and 8.49. However, we should not directly use these scores, since the scores count the potential speeds coming from both sides. By temporarily removing the edge between V_A and V_B, the mutual potential abilities of them become 0 and 6.62. As a result, V_B gets $(0.01 + 6.62) \times 1000/(1 + 0 + 0.01 + 6.62) \approx 869$ units of workload. Note that the computation of the potential speed is related to the size of work. Here, we present another simpler approach to compute the potential speed. After temporarily removing the edge between V_A and V_B, the potential speed of V_B can be estimated as $[\lambda_{BC} + (1 - \lambda_{BC})\lambda_{BD}\lambda_{DC}]S_C + [\lambda_{BD} + (1 - \lambda_{BD})\lambda_{BC}\lambda_{CD}]S_D + (\lambda_{BC}\lambda_{CE} + \lambda_{BD}\lambda_{DE} - \lambda_{BC}\lambda_{CE}\lambda_{BD}\lambda_{DE})S_E = 2.764$. By using this score, V_B gets $(0.01 + 2.764) \times 1000/(1 + 0 + 0.01 + 2.764) \approx 735$ units of workload.

3.5.7 Further Discussion

In reality, it is possible that a work has more than one owner. For example, a study group possesses a certain amount of data, which needs to be processed. Each member of the group could be the work's owner. In order to guarantee that the work segments that originated from different members have the same work identity, the group may choose the smallest identity of its group members, and concatenate it with the releasing time as the work's identity. For ease of description, we call such cases *multiple sources conditions*. We can assume that there is a virtual node, which only has contact with each of the group members. Since the workload allocation process at all nodes, in multiple sources conditions, is the same as that in the single source condition, our proposed adaptive workload allocation scheme can still function well. Essentially, the multiple sources conditions just shorten the length of the initial propagation phase. Since the work segments' diffusion patterns are similar to the patterns of a single source condition, we do not provide the illustration figures.

Any node could be the owner of a work. Therefore, it is possible that different works coexist in our system. Clearly, if there is always a large time gap between the arriving times of different works, our schemes definitely work. However, it is also possible that some works arrive while other works have not been completed yet. Clearly, the coming of new works may cause an unignorable difference between the recorded workload statuses of other nodes and their real conditions. However, since the goal of our chapter is to maximize the system's overall throughput, instead of minimizing each work's completion time, all of our discussed schemes do not need to discriminate the segments of different work.

Note that, when multiple works coexist in a system, minimizing the works' completion time becomes a very difficult problem. The segments from different works inevitably compete for the computing resources. Moreover, since social-crowdsourcing is a fully-distributed system, and the makespan of a work is determined by the finishing time of the last segment, it is very hard for a node to appropriately allocate the computing resources to different works based on the node's local view. One possible research direction is to explore the backpressure routing algorithms [40, 46, 47]. However, unlike the traditional routing problems, social-

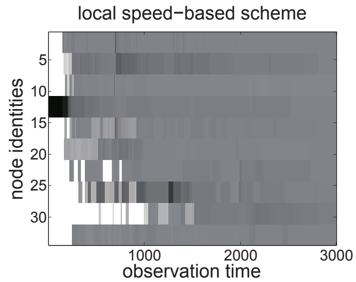


Figure 3.16: Multiple works: case 1.

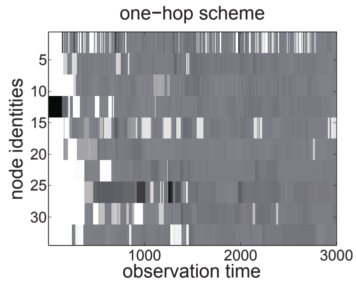


Figure 3.17: Multiple works: case 2.

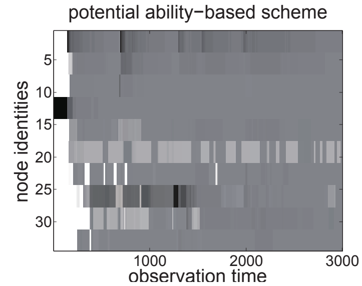


Figure 3.18: Multiple works: case 3.

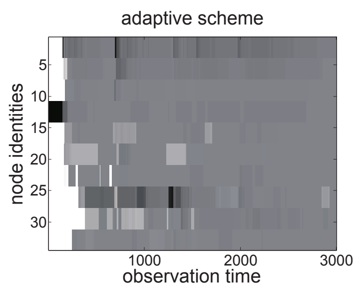


Figure 3.19: Multiple works: case 4.

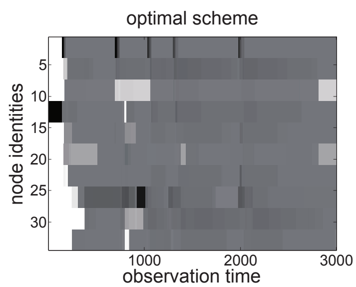


Figure 3.20: Multiple works: case 5.

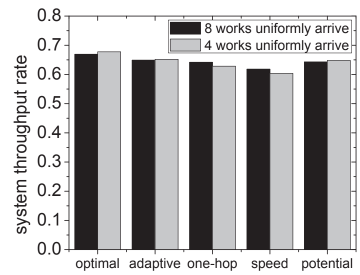


Figure 3.21: Throughput rate comparison.

crowdsourcing does not have the concept of message destination. We leave the problem for minimizing the completion time as our future work.

Figs. 3.16 to 3.20 illustrate the work segments' diffusion patterns under different work allocation approaches. During the simulation, we randomly create 4 works with a relatively large size. However, unlike the results in single source condition, the work's processing progression curves are closer to each other. In order to clearly observe the differences between schemes, instead of drawing the processing progression graph, we consider the system throughput rate by each scheme, as shown by Fig. 3.21. From the simulation results, we find that, comparing to other schemes, the coexistence of multiple works has a negative effect on the 1-hop information-based solution. One possible explanation is that, among all solutions, the 1-hop information-based solution highly depends on the recorded workloads of neighbor nodes; due to the arrival of new works, the workload status information changes much more frequently. When the changing rate exceeds the records' updating rate about a majority of neighbors, nodes will make wrong workload allocation decisions, which could also result in a chain reaction. Although the adaptive scheme also uses the historical information, it still has the best performance. The reason is that, whenever a node receives a statue of information indicating the existence of a new work, the node will mainly use the workload allocation decision based on potential ability. Therefore, the chain reaction of the wrong decisions is avoided.

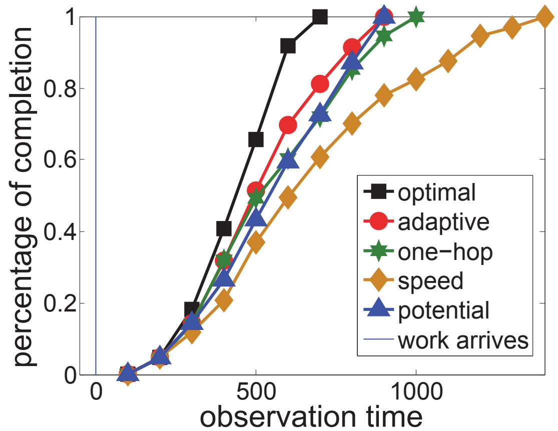
3.6 Performance Analysis and Evaluation

In this section, we conduct extensive simulations to evaluate the performances of our proposed algorithms. For ease of comparison, we call the naive scheme, which simply splits workloads according to the contactors' local computing speeds, *speed*, the allocation scheme based on each contact's local 1-hop historic information is called *one-hop*, the algorithm that pre-calculates a participant's future potential computing ability is called *potential*, and the algorithm that adaptively switches different allocation methods is called *adaptive*. In order to compare these solutions with the optimal results, after the whole observation is completed (posterior knowledge), we find the optimal allocation and use *optimal* to represent it. Based on the posterior knowledge, one can compute the shortest path from source to participants, and the optimal allocation can be found by letting all participants complete their workload at the same time. Although this algorithm is useless in practice, we can use its result as a comparison criterion.

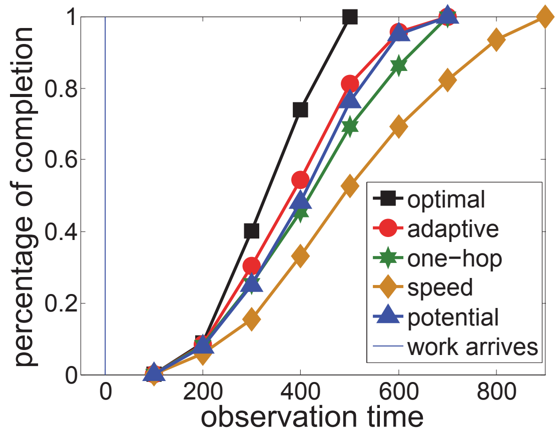
3.6.1 Evaluation Setup and Comparison Metric

The evaluations use both synthetic data and real data. In the synthetic simulation, we use a real social network dataset called karate club [42]. However, since in real life the social-crowdsourcing system always contains too much diversity, such as users' computing features, in order to quickly and accurately find out the changing trends of our interested attributes, in each simulation, we only change one variable. The work source nodes are randomly selected, and each simulation is executed 10 times. There are 34 users, and average inter-contacting time between each pair of nodes is uniformly distributed from 40 units of time to 100 units of time. The computing speed of nodes also follows a uniform distribution from 0.0001 to 10 units of work per unit of time. The total size of a work is set as $200,000$ units. The accepting rates are uniformly distributed in the interval from 0.35 to 0.99. We take $3,000$ units of time as the length of observation. In the real data-based experiments, we use Infocom06 trace. We eliminate the nodes with very low contacting frequencies, and we only use the data that was collected during the daytime.

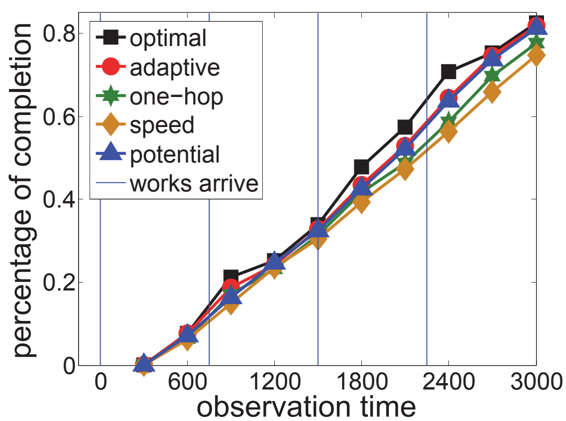
For efficient comparison, we adopt two metrics: percentage of completion and system (average) throughput rate. As we have mentioned in the introduction, the more nodes that join the system at an earlier phase, the sooner the work can be completed. The first metric can clearly show how well work segments are disseminated and processed within the systems at any time. By using this metric, one can observe the impacts of different methods on work segment diffusion. The second metric calculates the utilization of the system's available resources. The system throughput rate is computed as the ratio between the number of computing resources that have been used during an observation (the time interval before a majority of work segments has been completed) and the total number of available resources that the system can provide. Due to the existence of work segments' propagation delays, system



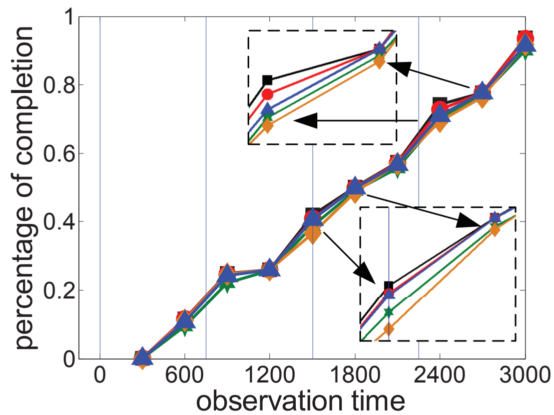
(a) single source, accepting rate: 50%



(b) single source, accepting rate: 90%



(c) multi-source, accepting rate: 50%



(d) multi-source, accepting rate: 90%

Figure 3.22: The impacts of average accept ratio on processing progress.

throughput rate can never reach 100%, even for the optimal result. The closer that a scheme comes to approaching the optimal result, the higher the system throughput rate is. The simulation consists of three parts. The first part focuses on the allocation of a single work on a regular social contacting graph; the second part shows the results regarding multiple works coexisting in our system, and the last part gives the results based on the real trace.

3.6.2 Evaluation Results

Fig. 3.22 shows the impacts of average accepting ratio on task's processing progress. During the simulation, we keep the average inter-contacting time of nodes uniformly distributed from 40 to 100 time units. The computing speeds are also randomly and uniformly generated with the interval $(0, 10]$ units of work per unit time. The minimum average acceptance rate is 35%, and the maximum is 99%. In Fig. 3.22, the x-axis represents the observation time, and the y-axis indicates the percentage of workloads that have been completed by the observed time. We can see that, with the growth of the accepting rate, the speed-based allocation scheme performs worse. One possible explanation is that when there are more participants in the system, the scheme cannot properly determine the workloads during each time of contacts. However, as the average accepting rate grows, it processes more quickly: the shape of its curve becomes closer to the upper left corner of the graph. For the other four methods, their work's processing progresses also become better at the high accepting rate scenario. Since the percentage of completion lines may become closer to each other, for the rest of evaluation, we focus on the average value of the system throughput rate.

Fig. 3.23 shows the impacts of average accepting ratio on the system throughput rate. With the increasing of the average accepting rate, all schemes' system throughput rates are growing upward. When the system has only a single task (Fig. 3.23 (a)), the increasing speed of the optimal scheme is much faster than the other four solutions, and therefore, there is a gap between the optimal result and others. However, when there are multiple tasks (Figs. 3.23 (b) and (c)), the optimal scheme, the adaptive scheme, and the potential-ability-based scheme have similar throughput rates at the higher accepting rates. One possible explanation is that, when there is a single work within a social-crowdsourcing system, a certain amount

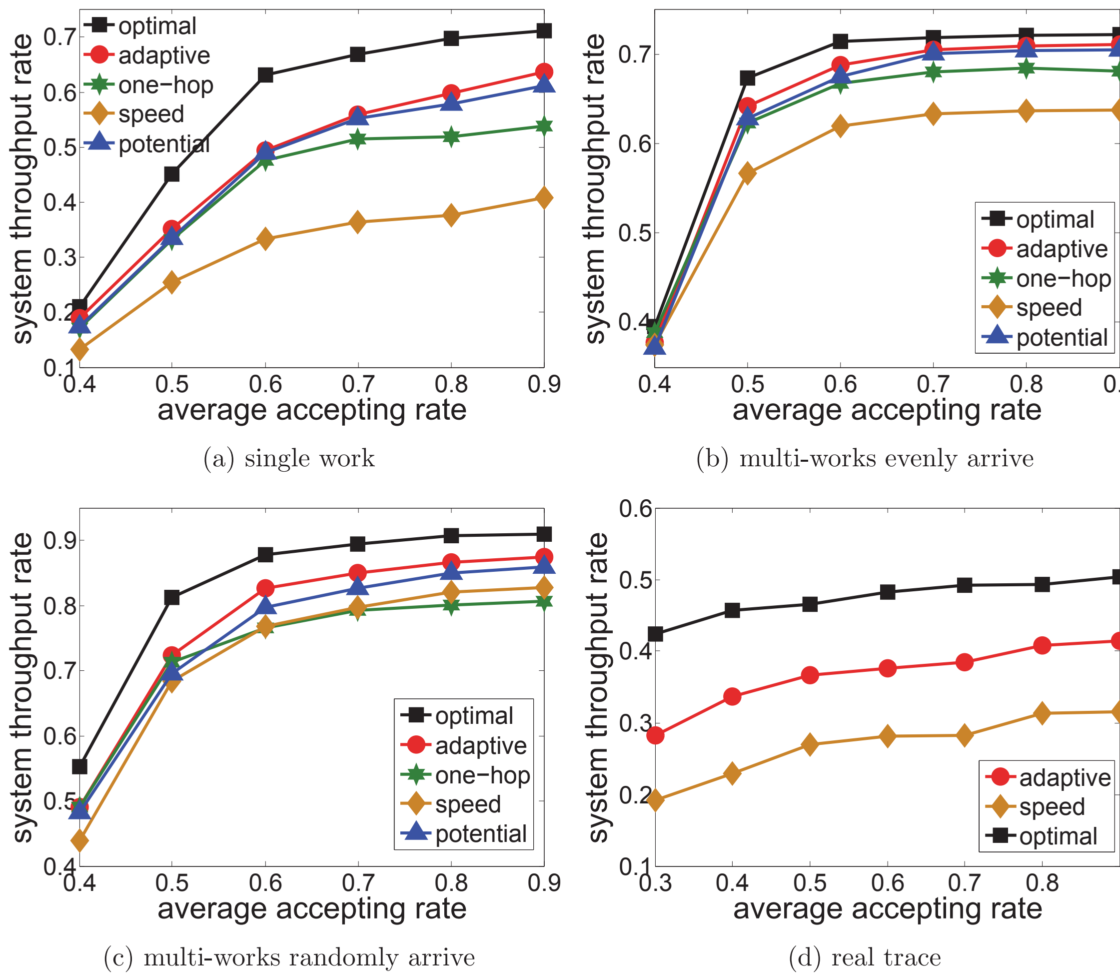
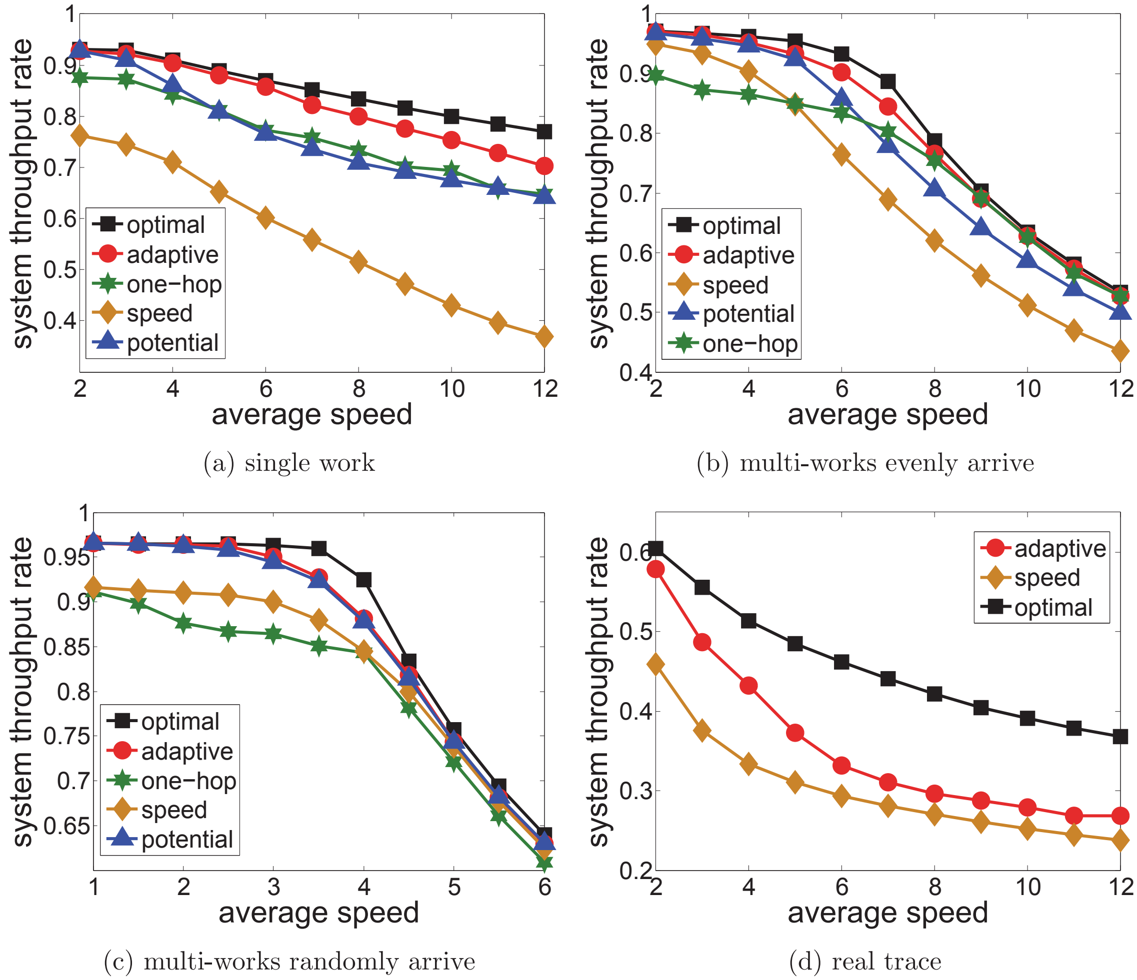


Figure 3.23: Impacts of average accepting ratio on the system throughput rate.

of computing resources is wasted due to the inappropriate workload assignments; however, when the system contains multiple works from different sources, those unused resources are taken by other works, and therefore, the system throughput lines become closer to the optimal one in multiple works condition. Fig. 3.23 (d) gives the simulation results on real trace. For Fig. 3.23 (d), we can see that the system throughput rate goes up with the increasing of accepting rate and becomes relatively stable at high accepting rate part. Note that the current version of our social-crowdsourcing system only considers the pairwise contacts, but in this real trace, there are situations where a group of users encounters each other at the same time. In these situations, we simply assign a sequential order to each pair of contacts, and then apply the workload allocation algorithms.

In Fig. 3.24, we check the impacts of users' average computing speeds on the system throughput rate. We let the average speed gradually change from a relatively low speed to a high speed. During this process, we check the system throughput rate under three distinct conditions: a system with a single work, a system with multiple works (periodically arrived), and a system with multiple randomly arrived works. We also test our scheme on real trace with a single work source. All simulation results show that, with the average speed growing upward, the throughput of all workload allocation methods decrease. When nodes have higher computing speeds, obviously, the completion time of the works can be reduced. As a result, the effect of the amount of wasted resources during the work segments' propagation phase exacerbates the system's overall throughput. We further check the influence of the average computing speed on the changing pattern of completion time. For a given fixed-size work, with the growing speed, the completion time becomes smaller, but the reducing speed is not linear. One possible explanation is that, with the growth of the speed, a task can be finished before other workers participate in the task. One interesting phenomenon in Fig. 3.24 is that, comparing to the situation of single work source, the performance of one-hop scheme becomes worse when multiple works coexisting in the system. Since the contacts in our system are intermittent, we think this phenomenon is related to the timeliness of historic neighborhood workload records. One way to solve the problem is to assign a time window for each node's historic records. The length of the window must be related to the node's contacting rates with others. For example, we can first set up a threshold l about contact times, and then compute the expected time Δt for a node to have l times of contacts with others. The expected time Δt will be used as the window's size. Once a record about historic workload expires, it will be invalid.

Next, we consider the influence of the size of works. In social-crowdsourcing, a



(a) single work

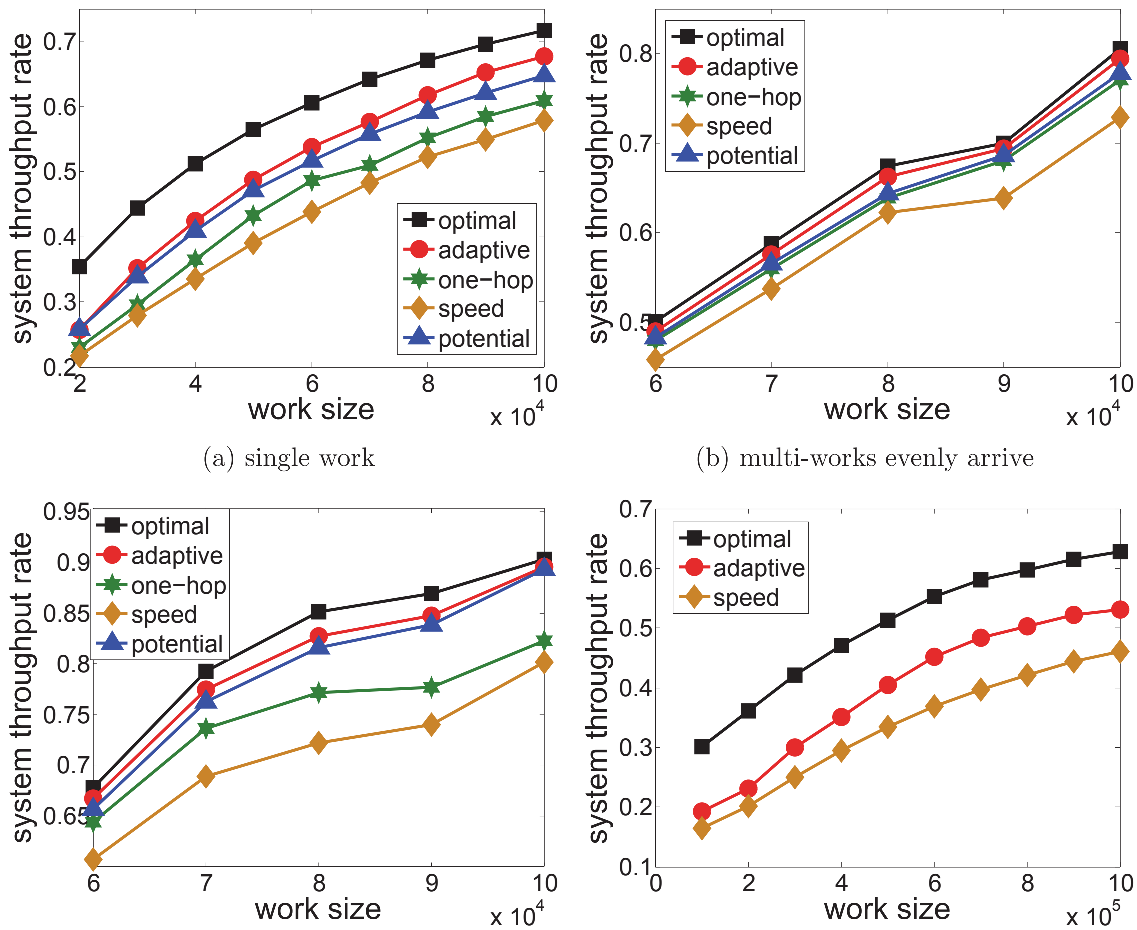
(b) multi-works evenly arrive

(c) multi-works randomly arrive

(d) real trace

Figure 3.24: Impacts of average speed on the system throughput rate.

system's computing resources are wasted during the segments' propagation phase. For a huge work, its work segments' propagation times are relatively small, compared to its completion time. Therefore, in general, the system throughput rate of a large work must be greater than that of a small-sized work. The simulation results are consistent with our qualitative analysis. In Fig. 3.25, the throughput rates increase along with growing work sizes. When there is only a single work in the system, there is a small gap on the throughput values of the optimal scheme and the others. However, when the system contains multiple works, such a gap disappears. The reason for this phenomenon is that, when nodes fully-occupy work segments (from plenty of tasks), the nodes always possess unfinished segments. Therefore, when propagating a task's



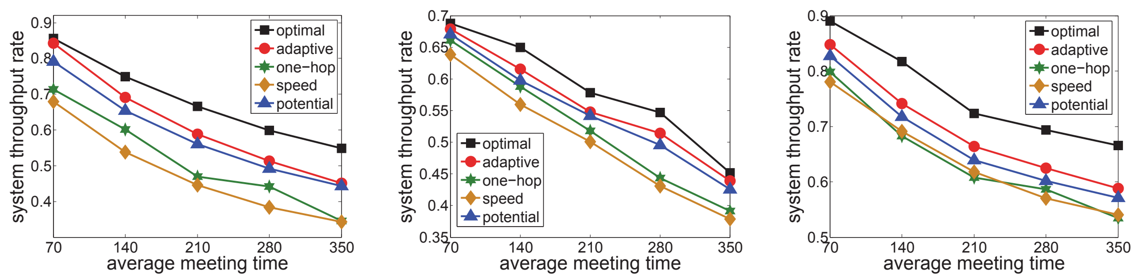
(a) single work (b) multi-works evenly arrive

(c) multi-works randomly arrive (d) real trace

Figure 3.25: Impacts of the size of works on the system throughput rate.

segments, workers are using their resources on another task; no system resources are wasted.

Fig. 3.26 shows the impacts of users' average inter-contacting time. The average inter-contacting time directly affects the frequency of the segments' redistribution among nodes. The more frequently segments are exchanged, the more likely it is that the workloads of workers are balanced. As shown by Fig. 3.26, all schemes' system throughput rates are decreasing with the growth of average inter-contacting time. Moreover, all of them have a similar decreasing rate. It seems that the average inter-contacting time has a similar influence on all workload allocation approaches.



(a) single work (b) multi-works evenly arrive (c) multi-works randomly arrive

Figure 3.26: Impacts of the average length of inter-contacting time on the system throughput rate.

Fig. 3.27 shows the impact of the number of work sources. In this simulation, we assume that there is a huge work, the segments of which are owned by several users. Unlike the condition of multiple works, here, the multiple source condition requires that all owners of a single work put the work into social-crowdsourcing at the same time. During the simulation, we gradually increase the number of the work segments' owners from 1 to 10. The total size of the whole work is a fixed value, and all work sources are randomly selected. From Fig. 3.27, we can see that our proposed adaptive scheme has the highest system throughput rate, compared to the other three approaches. All methods experience an increment with the growth of the number of sources. Since the total number of users is a fixed value, increasing the number of work owners can reduce its completion time, but the speed must converge to the condition that every node has already possessed his corresponding workload at the beginning.

When a social-crowdsourcing system contains multiple works, the arrival of a new work may disturb the current workload distributions. In this simulation, we want to check the mutual influence among works. Fig. 3.28 shows the impact of the number of coexisting works on our system, especially when the works randomly come at different times. If one work has not been finished while the other works appear, there must be certain interference between the old and the coming new works. In the simulation,

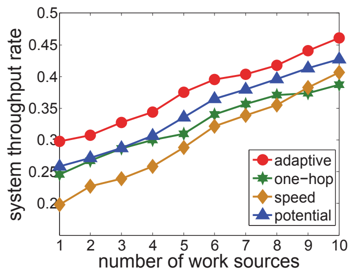


Figure 3.27: Impact: the number of sources.

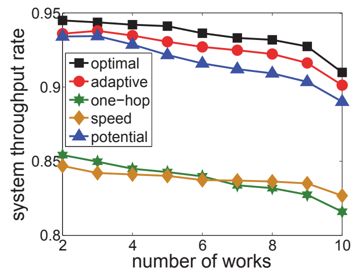


Figure 3.28: Impact: the number of works.

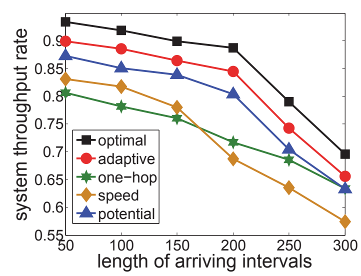


Figure 3.29: Impact: arriving density.

there are 2 to 10 works randomly coming during the observation time. The total length of the observation is set as $3,000$ units of time. The works' arrival times follow a uniform distribution. Also, the works' owners are randomly selected, and the total size of the works is a fixed value. We observe the system throughput rate decrease with the growth of the number of works. Moreover, all of them have a similar increasing pattern. Fig. 3.28 shows that the coexisting of multiple works do influence the workload assignments.

Since the arrival of a new work may interfere with the existing ones, in this part of the simulation, we check the impacts of works' arriving densities. We let works be periodically generated at some random nodes, and we gradually increase the period from 50 to 300 units of time, which means the arriving densities change from high to low. Fig. 3.29 gives simulation results: with the growth of the length of periods, the system throughput rate decreases. Although our approximation algorithms try to assign certain workloads to nodes such that they can complete the task at the same time, the real finishing times are different; In fact, lots of resources are wasted during a work's ending stage. When works arrive in a shorter period, the nodes, who just finished the previous work, are able to work on the new one.

3.7 Chapter Summary

In this chapter, we propose a new crowdsourcing system, called social-crowdsourcing. Unlike the traditional outsourcing platforms, our system is a distributed and self-organized system, which explores the social relations among users. Based on the proposed system, we consider the problem of workload partitioning and allocation among users. Unlike the conventional scheduling problems, the solution to our problem needs trade-offs between users' computing abilities, participating probabilities, and their social neighbors' capabilities. By observing the work segments' diffusion pattern, we find that, at different processing stages, we should adopt different workload allocation schemes, from the progressive one to the conservative one. To solve the problem, we first propose a conservative approach and a progressive approach, and then, we design an adaptive mechanism to combine the two approaches. The proposed progressive scheme is based on the overall computing ability of each user's future potential contacts, and the conservative scheme takes the historical workload statuses of nodes' 1-hop neighbors into account. For automatically and gradually switching between different approaches, we consider the distribution of each node's expected completion time, based on collected historical information: a highly biased distribution indicates that the system needs a progressive way to transfer the surplus workloads to others, while an approximately uniform distribution simply requires local adjustment. Our solution is applicable, not only when a system has a single work, but also when the system contains multiple coexisting works. Extensive simulations prove the significant performance of our schemes.

Here are the main contributions of this chapter:

1. We create a distributed and self-organized crowdsourcing scheme within mobile social networks called *social-crowdsourcing*.

2. Due to the unique features of social-crowdsourcing, we find a new research

topic: the optimal workload assignment problem in intermittently connected distributed system.

3. Based on social information and other auxiliary knowledge, we propose a set of workload allocation algorithms, from the most conservative way to the most progressive way.

4. We provide extensive simulations to show that our new system can provide a better decision on the workload assignment problem.

CHAPTER 4

EXPLORING SOCIAL ENCOUNTERS IN THE DESIGN OF COOPERATIVE TRAJECTORY MAP-BASED NAVIGATION

In the previous two chapters, we have shown how social information can be directly used in the decision-making process, such as the organizing structure of a set of P2P nodes and the amount of assigned workload on a node in social-crowdsourcing. However, there are some other applications, such as Facebook and Cooperative trajectory mapping, which are built over the actual social relations, instead of explicitly manipulating certain social features. In this chapter, we show how these applications benefit from the social information. More specifically, we focus on cooperative trajectory mapping and show that, by carefully exploring users' social encounter records, the quality of its navigation result can be improved.

4.1 Introduction

Maps are useful in helping people navigate through unfamiliar places. However, ensuring that maps remain up-to-date is a constant challenge. Moreover, when friends are exploring some new places, electronic maps may not always be available, which thus motivates researchers to consistently develop more accurate and efficient map construction techniques. Cooperative trajectory mapping is an emerging technique for map construction that takes advantage of the different sensors that are embedded in smartphones to create maps of a group of friends' trajectories. This type of map

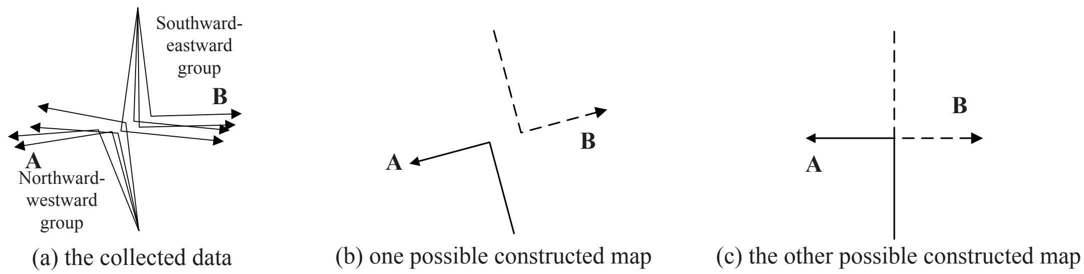


Figure 4.1: The effect of a measurement error.

is known as a *trajectory map*.

Unlike traditional map construction problems, GPS is not used when cooperatively building the map due to its high energy consumption [48, 49] and the unavailability of GPS signals in certain environments, such as indoors. Instead, the smartphone's sensors, like the accelerometer and electronic compass, are used to collect information like the departure distance, moving speed, and direction between consecutive sampling times [48]. This data is then transmitted to a central depository via a 3G or 4G connection, which, in turn, processes the data from multiple users to construct a trajectory map. Due to the concerns about location privacy and security, the cooperative trajectory mapping is mainly used by a group of mutually trusted users, such as friends. The trajectory mapping technique can also be used in various applications, such as traffic monitoring [50], public transportation tracking [51, 52, 53], and localization [49, 54, 55].

An important issue that arises when constructing a trajectory map is dealing with measurement errors from the sensor data. The measurement error is also known as measurement noise, or noise. A slight measurement error can have a larger impact on the overall map if left uncorrected. For example, a 1° error on a compass will result in a difference of 1.74 miles after a user has traveled 100 miles. Prior researchers have also recognized the importance of measurement errors, but to date, research has only used a simple noise model to address the problem [48].

We illustrate the impact of error correction in the example shown in Fig. 4.1. Suppose that there are two groups of users; one group moves North and then turns West, while the other group moves South then turns East. Users in each respective group walk along the same path. In Fig. 4.1(a), we use solid arrows to depict each user's reported trajectories. Due to the effects of measurement noise, there is some variance in the reported trajectories, even though all of the users are traveling along the same path. As a result, based the collected data, the central server may build either the map as shown in Fig. 4.1(b), or the one depicted in Fig. 4.1(c). We cannot simply average the data to build the map since it is possible that the noise may not follow a normal distribution. During the constructing process of a trajectory map, two spatially disjointed paths (or trajectories) may be falsely reported as a pair of paths intersecting with each other (*false positive*), or two joined paths may be depicted as unrelated (*false negative*). Moreover, when the sensor devices contain systematic measuring errors, the accuracy of a constructed map will be significantly influenced.

We propose a social encounter-based error cancellation system, which can provide correct routing results on the trajectory map, even if every phone has different error properties. The main idea of our approach is to cancel the relative errors among participators; although the absolute error of participators' devices may be large and hard to correct, their relative errors can be easily detected, measured, and corrected by multiple times of physical encounters. For a group of users, whose devices do not have the relative error, better routing results can be found by exploring spatial intersections of their trajectories.

4.2 Literature Review

One of the earliest applications of cooperative trajectory mapping is a mobile social network-based [56] navigation system that was proposed by Constandache et al. [48].

Each user in the mobile social network will periodically report his trajectory and his encounter information to the server. The server will use this information to build a set of directions and displacements, which allows friends to locate each other. Later work by Constandache et al. [57], and Thiagarajan et al. [58, 49] also applied a similar idea to other applications. The main difference of our work is that prior research used a relatively simple noise model and only considered noise cancellation by a single user, while we consider a more general noise model that has both systematic errors and random noise. We use encounter information among multiple users to reduce the relative errors, and further, provide better routing results.

In paper [59], the authors propose a low-energy and accurate trajectory mapping approach, and they discuss the trade-off between GPS and multi-sensor devices. However, their work is based on the assumption that an electronic map is available to use. In our problem, based on users' uploaded data, the central server needs to build a trajectory map, on which users can be localized and navigated.

Cooperative trajectory mapping shares similar characteristics with the inertial navigation system (INS) that is used in submarine navigation [60, 61, 62]. Both techniques use the recorded moving distance and direction to determine a location, and both are subject to drifting because of the sensors' noise [63]. INS research has two general approaches to address this problem. The first approach is to use filtering techniques, such as the Kalman filter [64, 65] and particle filters [66, 67], to limit the effects of the measurement noise. The second approach is to apply noise cancellation methods using GPS, assisted GPS, or Wi-Fi [54, 55, 68, 69]. A key difference in our approach is that our technique is more flexible since we emphasize the related locations of each user rather than the physical locations. Our approach still works even if there is no external location reference (also known as an anchor), such as WiFi Access Point (AP).

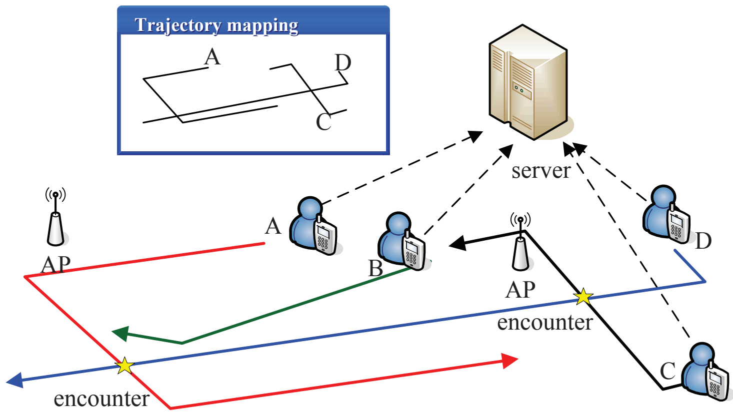


Figure 4.2: System model.

Priyantha et al. [70] propose an anchor-free localization (AFL) algorithm to resolve the localization problem in sensor networks. The goal of [70] is to determine the position coordinates of every sensor via local node-to-node distance, even if the physical location of the nodes is unavailable. However, this solution cannot be used to build a trajectory map because the positions are static spot locations. In the process of creating the trajectory map, we consider the trajectories of moving nodes.

4.3 Background

4.3.1 System Model

A typical cooperative trajectory mapping system has two basic components, as shown in Fig. 4.2.

1. **A central server.** The server collects users' trajectory data and builds a trajectory map. The server also provides additional services based on the constructing map, such as friend locator or location routing.

2. **User smartphone.** Each user reports its trajectory and encounter information to the server. The reported trajectory is recorded as a set of displacements and directions between consecutive samplings.

Besides these two building blocks, our approach takes the advantage of any AP, such as a WiFi AP, which a user encounters. An AP serves as a fixed location reference, and the physical location of the AP does not need to be known. The purpose of the AP is to quickly establish the spatial relationship among each user's movement trajectory and to provide an external global reference for noise cancellation. The AP periodically broadcasts time-stamped beacons, and when a user receives the beacon, he records this encounter event and reports it to the server. The broadcasting of the beacon is part of the 802.11 standards; any WiFi AP can be used.

We assume that there are N participants in this system. Each user's mobile phone is equipped with an accelerometer, a compass, a wireless receiver, and an encounter sensor. The accelerometer and compass are used to determine a user's displacement and direction, respectively. The wireless receiver is used to receive beacons that are transmitted from the AP. The encounter sensor is used to periodically signal and record the presence of other users. This can be accomplished by using a Bluetooth module built into the smartphone [48].

A user's smartphone maintains two lists, a *movement list* and an *encounter list*. The smartphone will periodically report the two lists to the server via a 3G/4G connection [71]. The movement list consists of displacements and the moving direction from the last recorded position. The encounter list consists of user IDs and time stamps that denote when the encounter occurred. We use mathematical $0°$ to represent East and $180°$ to represent West. The position of a user at time t can be computed by:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} + L \times \begin{pmatrix} \cos(\Theta_t) \\ \sin(\Theta_t) \end{pmatrix}, \tag{4.3-1}$$

Table 4.1: Table of notation

L	Reported displacement
$k(t)$	Systematic error in displacement measurement
l	Real displacement
N	Number of participants in the system
λ	Accelerometer's random noise
Θ	Reported moving direction
θ	Real moving direction
$\Delta\theta(t)$	Electronic compass systematic error parameter
δ	Compass's random noise
T	Cycle time for reporting data to the server
x_t, y_t	The coordinates of a user's position at time t
$\overrightarrow{E}$	Accumulated error
$\overrightarrow{V}$	Adjustment vector

where L is the reported displacement between two measurements, and Θ is the moving direction. All of the symbols used in this chapter can be found in Table. 4.1. Note that the trajectory of each user is recorded in his own coordinate system, which is only relative to the initial (unknown) location of the user [48].

The server will use the two lists from each user to derive a trajectory map. If each measurement is accurate, the users' position information at each time is fully preserved by the server.

4.3.2 Existing Noise Cancellation Solutions

The general idea behind error cancellation in prior work is that each user's noise can be corrected by some physical references [48]. If a user passes by the AP (the user is in the communication range of the AP), the server can then compute the amount of accumulated errors, which causes the trajectories to drift; then, the user's trajectory can be repositioned. Let the detected *accumulated error* be $\overrightarrow{E}$. The corresponding *adjustment vector*, which is generated by the server for error cancellation, is $\overrightarrow{V}$ ($\overrightarrow{V} = -\overrightarrow{E}$). If user A encounters user B, who has just been repositioned with the help of an AP, the trajectory of A can also be corrected since the position of B

is likely to be more accurate. Consider that the amount of noise is proportional to time [48] if each user moves with a constant speed. Paper [48] also proportionally uses the instantaneous correcting vector to adjust the historical trajectory. For example, let t_1 be the previous time for adjustment; the server detects a new adjustment vector $\overrightarrow{V}(t_2)$ at t_2. Then, the adjustment vector of the position at time t ($t_1 < t < t_2$) is given as:

$$\overrightarrow{V}(t) = \frac{t - t_1}{t_2 - t_1}\overrightarrow{V}(t_2). \tag{4.3-2}$$

The solution is inadequate due to the following reasons. Firstly, the direction of the adjustment vector used is the same in all of the data. However, when there is a systematic error of compasses, we should use rotation to adjust the trajectory, which means the direction of the adjustment vector should be different at different data points. Secondly, the existing solution essentially depends on APs, and only false negative encounters are used in the error cancellation; false positive encounters, which also contain certain information, are ignored, and the existing error cancellation approach does not work if APs are not available. By exploring the relative errors and false encounters, we can improve the error cancellation in trajectory mapping, and further provide better navigation results based on the constructed map.

4.3.3 Challenges

In order to correct measurement errors, we need to solve the following three issues.

Firstly, since each user' accelerometer and compass may exhibit different error parameters, it is difficult to estimate the absolute error parameters of each user, especially when physical references, such as APs, are not available. Without knowing these parameters, we are not able to correct the reported trajectories since we cannot determine the extent of adjustments. For example, in Fig. 4.3 (a), there are two users' reported trajectories, and both of them contain measurement error. Suppose that the

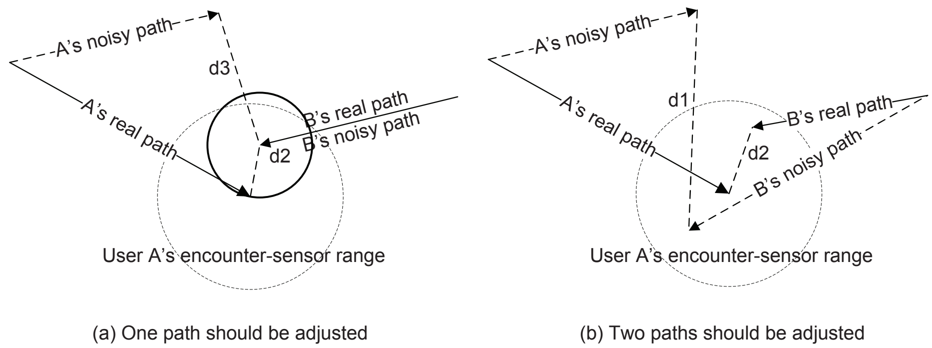


Figure 4.3: The challenge of making adjustments without the auxiliary information.

server can compute the spatial distance d_1 of the reported trajectories and can obtain the real distance d_2 from the reported encounter information; we cannot determine how much of an adjustment should be made and in which direction. Even if only one user is inaccurate, without knowing the identity of the accurate user, a correct adjustment is still hard, as shown in Fig. 4.3 (b). Considering that the accuracy of navigation mainly depends on the correctness of users' relative locations, we aim to eliminate the relative errors among participants, and further explore the spatial trajectory intersections of users, whose relative errors are zero, during navigation.

Secondly, there are two types of encounter errors which can be detected, and they should be treated differently. *False positive* means that two physically disjointed trajectories are falsely reported as a pair of intersecting trajectories, while *false negative* represents the situation where two physically joined trajectories are depicted as unrelated. In the false negative case, the server can obtain the actual adjustment distance and direction between their reported trajectories. However, in the false positive case, the server cannot attain them since the user is out of the signal range of an AP or another user's phone. In our system, we use false negative events to estimate the relative errors among users, and adjust their reported trajectory accordingly, while false positive cases are temporarily used to eliminate some obviously wrong intersections on a visual map.

Last, two reported trajectories having a spatial intersection without any physical encounter may not be a false positive event since the users may pass the intersection at a different time. Hence, speed restrictions and time should also be included for detecting the false positive, and our system only considers the reported trajectories which definitely will have an encounter.

4.4 Solution Framework

A key feature of our solution is that when two users meet, each user will independently report their encounter with the other to the server. Hence, we can use encounter information to adjust users' trajectories. Another feature of our solution is about relative measurement errors: Although the absolute error of devices may be large and hard to detect, the relative errors can be easily detected by multiple times of physical encounters. If we adjust users' trajectories such that their relative errors are so small as to be negligible, the cooperative trajectory mapping system can work well: the relative positions of trajectories' spatial intersections are correct. In a nutshell, our method exploits the relative errors and physical encounters of users, through which different users can interpret the physical world in the same way. As a result, instead of estimating absolute error parameters, the relative errors are considered.

We illustrate how the features are used in our solution by using the following example. Consider user A who is physically at location loc_α; however, the server may interpret user A's reported data as A is at loc_β because of the measurement error.

Without some physical references, we cannot detect the error or measure the magnitude of the error. However, if another user B, who encountered with A before, comes across A again, we can compute the relative error magnitude between A and B by joining their trajectories at the place, where they first met, and measuring the deviation angle and distance between their current locations. By fixing either

A or B's reported trajectory and adjusting (i.e. rotate and stretch) the other one's previously reported trajectories, the server eventually can build a map, where the relative locations of A and B are correct. Based on our error model, the systematic error parameters can be regard as fixed ones since they change very slowly according to time. By this way, if a group of users encounters each other multiple times in a period, our system can estimate their relative errors, adjust their trajectories, and eventually get a map without relative errors.

Our proposed method consists of three steps to reduce error:

1. At each smartphone, the application applies the Kalman filter to eliminate random noise. Considering that the Kalman filter only requires an individual user's historical moving pattern, rather than global information, it is more efficient to apply the Kalman filter on the user side. After filtering, users report their moving trajectories and encounter information to the central server;

2. The central server first coarsely adjusts the reported data based on previously estimated relative error parameters, then it detects new false positive cases and false negative cases. Next, the server adjusts users' trajectories to eliminate the inconsistency. During the adjustment, the server may make some hypothesis about the direction of corrections in false positive conditions, and a hypothesis is verified/removed based on the upcoming encounter events;

3. After correcting each user's trajectory, the server computes the relative error parameters among users, and it refines the estimated parameters based on new adjusted results.

When there is no AP present, steps (2) and (3) are carried out by our proposed anchor-free error reducing algorithm (AFER); when there are APs present, steps (2) and (3) are our proposed accessorial anchor-based error reducing algorithm (AAER).

4.4.1 Accessorial Anchor-based Error Reducing Algorithm

Algorithm 9 The AAER algorithm

```
1: for Each sampling time T do
2:     Find of false positive and false negative by mutual encounter and APs
3:     Use HBMS algorithm to adjust the reported trajectories
4:     Record the adjusted positions and relative error parameters
```

When APs are available, the chance of having encounters is increased. When users pass the same AP at different times, the corresponding positions on their trajectories should be joined at the same spot, which seems like users virtually encounter with each other. These encounter events can be used for estimating the relative error parameters. The procedure of AAER is shown by Algorithm 9. The details of line 2 are given by Chapter 4.5.2. The relative position adjusting process is like a complex spring-mass system in physics; we want to relatively adjust users' reported trajectories such that the relative error between different trajectories becomes zero. When eliminating relative errors, each point on a reported trajectory may be adjusted according to different adjustment vectors, which seems like a single spot in a spring system being pushed and pulled by adjustment forces from different sources. At each time step, after collecting all of the trajectories from users, the server runs a hypothesis-based mass-spring (HBMS) adjustment algorithm, Algorithm 10, to estimate users' actual relative positions. The HBMS algorithm is discussed in Chapter 4.5.3, and we use *adjustment force* $\overrightarrow{F}$ to represent the adjustment vector being used during a position adjustment.

There are two types of adjustment force: false positive-based adjustment force and false negative-based adjustment force. Since the adjustment direction in false positive is uncertain, we use two hypotheses to temporarily store the possible adjustment positions. Later, we use the upcoming encounter information to verify the hypotheses and to eliminate the wrong ones.

Algorithm 10 The HBMS algorithm

```
Verify previous direction hypothesis by current encounter
Adjust by new false negative and estimated relative errors
Set up new false positive direction hypothesis
for Each hypothesis do
    Compute adjustment force and adjust trajectories
    if Find a new false positive case from historic data then
        Set up new false positive direction hypothesis
Update current error parameters
```

4.4.2 Anchor-free Error Reducing Algorithm

When there are no APs present, we use anchor-free error reducing algorithm (AFER) to correct the errors. Consider that if all of the users have the same absolute error parameters, the cooperative trajectory mapping system can still be used well even if there are huge differences comparing to the real world; if the majority of the users have same error, we just need to adjust the other users by changing their data according to the majority of the users. Hence, our goal is to find out the relative errors among users based on their multiple times of encounters. Considering that the absolute systematic error parameters could slightly change according to time, we further use the adjustment force to refine the estimated relative error parameters and to eliminate the newly detected inconsistency.

Algorithm 11 The AFER algorithm

```
for Each sampling time T do
    Find out false positive and false negative error by mutual encounter
    Use the HBMS algorithm to adjust the reported trajectories
    Record the adjusted positions and relative error parameters
```

AFER is given by Algorithm 11. The difference between Algorithms 9 and 11 is that AFER does not use any AP-based encounter information. Unlike AAER, we assume that there is an error-free random walker and that the identity of this user is known by the central server. The reason for having this assumption is that in order to avoid tortuosity of the constructing map we need to find some physical references. Consider that there are several users moving in a large region by following some moving patterns. Without the physical references, it is possible that the final constructed map is contorted while sub-regions' maps are relatively accurate.

4.5 Algorithm Details

4.5.1 Error Model

The accelerometer and compass each have their own respective error model. Table 4.1 contains the notations used.

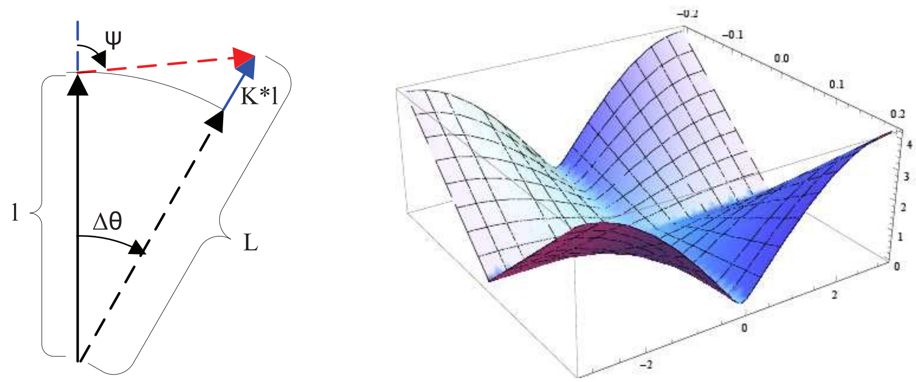
We first consider the accelerometer. There are two types of errors: the systematic errors and the random errors (i.e. white noise). The systematic error is proportional to the moving time or moving distance. Moreover, the magnitude of the systematic error may change over time. For example, a person walks continuously for a couple of hours. The size of his average step at the beginning will be different from that towards the end. We use $k(t)$ to represent a systematic error which may slightly change over a long period of time. We let λ denote the random white noise, which follows the normal distribution. L represents the reported displacement, and l is the real displacement.

$$L = l + k(t) \times l + \lambda \tag{4.5-3}$$

The readings from an electronic compass, Θ, can be regarded as the sum of real data (θ), random noise (δ), and systematic error ($\Delta\theta$), which may also slightly change with the growth of time.

$$\Theta = \theta + \Delta\theta(t) + \delta \tag{4.5-4}$$

To demonstrate the effect of systematic error, we temporarily ignore the random



(a) Sensor error　　　　(b) Error change pattern

Figure 4.4: Error model.

noise. Let $p = k * l + l$. The measurement errors, as shown in Fig. 4.4 (a), can be computed as follows:

$$\text{The amount of } \overrightarrow{E}\text{: } |\overrightarrow{E}| = l \times \sqrt{p^2 - 2p\cos(\Delta\theta) + 1} \qquad (4.5\text{-}5)$$

$$\text{The direction of } \psi\text{: } \cos(\psi) = \frac{p\cos(\Delta\theta) - 1}{\sqrt{p^2 - 2p\cos(\Delta\theta) + 1}} \qquad (4.5\text{-}6)$$

In Fig. 4.4 (b), we use simulations to generate the change pattern of $\overrightarrow{E}$'s magnitude, where $l = 1$, $\Delta\theta$ varies from $-\pi$ to π, and k changes from -0.2 to 0.2. If one of the noise parameters is relatively large, both types of error cannot be ignored.

The random noise, λ and δ, can be eliminated by letting each user's smartphone apply the Kalman filter to process the data before uploading to the server. This can be accomplished since the random noise follows the normal distribution, and the moving pattern of a user can be computed by the user's smartphone itself. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means for estimating the state of a process in a way that minimizes the mean of the squared error [64].

Table 4.2: Table of notations for auxiliary functions

S_{max}	the maximum speed
(x_b, y_b) / (x_e, y_e)	beginning or end location of a displacement
t_b / t_e	beginning or end time of a displacement
d	length of a displacement
$(x(A,t), y(A,t))$	the location of user A at time t
R	sensing range

4.5.2 False Positive and False Negative Error Detection

At each reporting time, the server obtains users' reported positions and encounter information (When the actual distance between nearby users is less than device's signal radius, both users will have an encounter record.) In order to detect measurement errors, the server needs to find out inconsistency by comparing two records: the encounter information and trajectories. For the false negative error, we can compute the relative error between two users' devices by joining their trajectories at the place, where they previously met, and measuring the deviation angle and distance between their current locations. A relative error vector can be derived based on these two values. For the false positive error, the actual distance between users' current locations is unknown since they are not in the sensor range of each other. Here, we use the sensor radius to represent this distance. In practice, as long as the false positive-based adjustment can temporarily remove the detected inconsistency from a visual map, it could be any distance greater than the radius.

Having two spatially intersected trajectories on a visual map without the corresponding encounter record does not sufficiently indicate a false positive error since users may pass the same location at different times. Due to the fact that the instantaneous velocity of a user may vary, we should consider all of the possible moving conditions of a user when detecting false positive cases. In order to simplify the solution, we add a time dimension to the traditional X-Y coordinates. Table 4.2 contains the notations used in this subsection.

Given a specific distance, there are multiple ways in which a user can move. For instance, the user can first move at his maximum speed to finish the reported displacement and then stop and wait at the end. Alternatively, the user can also wait first at the beginning and then move to complete the distance just on time. Therefore, there are two trajectory boundary functions:

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - (t_e - d/S_{max})}{d/S_{max}} \tag{4.5-7}$$

$$\frac{x - x_b}{x_e - x_b} = \frac{y - y_b}{y_e - y_b} = \frac{t - t_b}{d/S_{max}} \tag{4.5-8}$$

Assuming that users A and B both report one displacement in a time interval from t_b to t_e. The initial position of A is $(x(A, t_b), y(A, t_b))$, and the end of the displacement is $(x(A, t_e), y(A, t_e))$. Similarity, we have B's displacement from $(x(B, t_b), y(B, t_b))$ to $(x(B, t_e), y(B, t_e))$. At a given time t, users A and B definitely encounter with each other if their reported trajectories satisfy the following formula:

$$(x(A, t) - x(B, t))^2 + (y(A, t) - y(B, t))^2 \leq R^2 \tag{4.5-9}$$

After simplification, we can get:

$$R^2(a^2 + c^2) - (ad - bc)^2 \geq 0 \tag{4.5-10}$$

where,

$$a = \frac{(x(A, t_e) - x(A, t_b)) - (x(B, t_e) - x(B, t_b))}{t_e - t_b} \tag{4.5-11}$$

$$b = -a \times t_b + x(A, t_b) - x(B, t_b) \tag{4.5-12}$$

$$c = \frac{(y(A, t_e) - y(A, t_b)) - (y(B, t_e) - y(B, t_b))}{t_e - t_b} \tag{4.5-13}$$

$$d = -c \times t_b + y(A, t_b) - y(B, t_b) \tag{4.5-14}$$

Hence, a pair of spatial intersected trajectories without any corresponding encounter record may or may not be the false positive case. In our solution, the false positive only refers to the cases, where the reported data indicates definitely encountered while having no corresponding encounter records. Clearly, the probability of having a false positive case is much less than that of having a false negative case.

4.5.3 Hypothesis-based Mass-spring Adjustment

Hypothesis-Based Mass-Spring adjustment (HBMS) is used to estimate the optimal relative positions of users. HBMS first computes the adjustment force in false positive and false negative cases, respectively. Since the adjustment direction of false positive is unknown, the HBMS algorithm makes hypotheses about the direction, and then, based on the hypothesis, it recursively repositions the locations of other users, the relative error with whom has not yet been detected. Wrong hypotheses will be eliminated later in hypothesis verification. For the incoming data, our system first uses some relative error parameters, which have been found previously, to make a coarse adjustment, and then, it further refine the positions of users if any new inconsistency is detected. After finding the optimal relative positions, HBMS will update the error parameters of users. With the growth of time, eventually, the relative error parameters among all users can be found, and the reported trajectories can be adjusted accordingly. The final cooperative trajectory map is free of relative errors.

4.5.3.1 Adjustment Force

Assume that there are two users, i and j, who are in the sensing range of each other's device, and that they have encountered each other before. In a false negative case, after joining their trajectories at the place, where the last encounter happened, their current locations are disjoint. Note that, if there is no relative error, their current locations should be the same since they just come across each other again. We can compute the distance $\widetilde{d_{ij}}$ between users' current locations on the constructing map, and we can also obtain the real physical distance d_{ij} through RSSI readings. The adjustment force $\overrightarrow{F_{ij}^{-}}$ can be calculated as:

$$\overrightarrow{F_{ij}^{-}} = \overrightarrow{u} \times (\widetilde{d_{ij}} - d_{ij}), \tag{4.5-15}$$

where $\overrightarrow{u}$ is the unit vector from user i's location to that of user j. Note that, in practice, since the radius of encounter sensors is much smaller that the length of a road stretch, when two users encounter, we usually treat their real physical distance as zero. Since the trajectories of users are relatively adjusted in our approach, the server only needs to hold one user's trajectory and adjusts the other. The adjustment forces are associated with the users, whose trajectories have not been updated for a longer time or have only been adjusted according to a smaller portion of users' trajectories. In addition, if there are some other users, who have already been adjusted and have zero relative error with user i (or j), their trajectories should also be adjusted in the same way as i's trajectory (or j's.)

In the false positive case, we cannot obtain the real distance d_{ij} or the adjustment direction. As shown in Fig. 4.5, the real path can be located at either the same side of the error path or the other side. Therefore, we need two hypotheses to respectively store the adjustments.

$$\overrightarrow{F_{ij}^{+}} = \pm\overrightarrow{u} \times (R - d_{ij}) \tag{4.5-16}$$

The false positive cases are much harder to detect than false negative cases; the amount of false positive-caused adjustment force is much less than that of false negative. Note that our system cannot guarantee the accuracy of the false positive adjustment; what we do just temporarily eliminates some obvious errors but keeps trajectories close to their originally reported places during the process of map

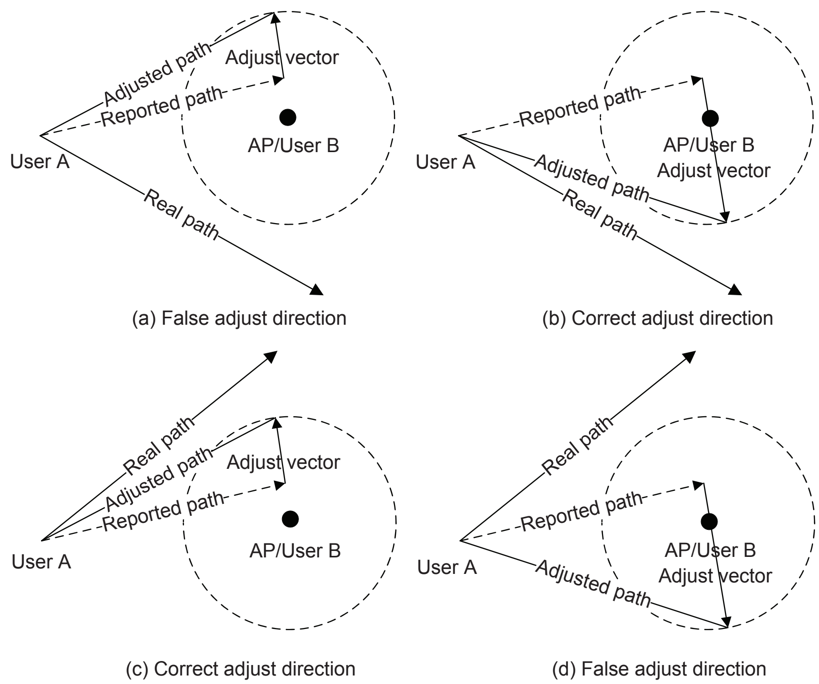


Figure 4.5: The uncertainty of adjustment direction in false positive cases.

construction. The final positions of adjusted trajectories are affected by both false positive and false negative adjustment forces.

4.5.3.2 Recursively Reposition

Changing one user's trajectory may also impact the trajectories of others. In other words, a false positive adjustment at time t may cause the happening of another false positive case on historic trajectories. As a result, the estimated position adjustment should be accomplished recursively: once a new false positive been detected in the historic data, new hypotheses will be generated to eliminate the errors.

4.5.3.3 Hypothesis Verification

A hypothesis can be verified by using follow-up encounters with other users or APs. Note that, if the corresponding user of a hypothesis involves in any newly detected false negative event, the user's trajectory will be adjusted according to the event. Since our system adjusts the relative positions of users, if the trajectory of a hypothesis always makes correct encounters, eventually, the hypothesis is kept since the corresponding trajectory has no relative error with other trajectories; if it incurs too many false encounters at a later time or cannot be verified for a long time, the hypothesis will be eliminated. Although using an AP is not a prerequisite in the HBMS, using APs will allow us to easily verify a hypothesis and estimate error parameters. When APs are not allowed, our system temporarily uses static users as virtual APs.

4.5.3.4 Error Parameter Estimation

In our problem, there are two phases of adjustment: before and after all users' relative error parameters are determined. Initially, all relative error parameters are unknown. Based on multiple times of physical encounters, our system estimates these parameters. For example, assume that the central server finds out that users A and B encountered each other twice in a period of time, then B's relative error parameters (with respect to A) can be estimated. If there is another user C, who came across each of A and B once, the relative error parameters of C can also be computed after removing the relative errors from the trajectories of A and B.

However, since the systematic error may change with time, the estimated relative error parameter at current time may not be accurate in future. Hence, the estimated relative error parameters should be updated after the first phase of adjustments. At the beginning of each time step's adjustment, our system uses the previously estimated relative error parameters to adjust the trajectories coarsely in advance, and then it applies HBMS. Based on the time interval between the nodes' (involved in the false encounter) previous adjustment times and current times, we calculate the ratio of the adjusted amounts to the corresponding time interval and update the estimated parameters.

4.5.4 Historical Error Cancellation

Users may not have any encounter in a period of time, and therefore, the reported trajectory in this period may be inaccurate. Once we know how to adjust the instant position, the historical positions can also be corrected. We name the process of applying error cancellation to the historically reported positions as historical error cancellation.

Assume that the server finds out an adjustment vector at time t. The existing solution [48] is to reposition the historical trajectories by using a proportioned adjustment vector. However, this adjustment is not true if the compass contains systematic errors. In Fig. 4.6, suppose that the systematic error for a compass is $\pi/4$, which means that moving directly north should be reported as moving towards north-east. If we use the existing cancellation algorithm, the adjustment vector in the middle will be parallel to the instantaneous adjustment vector, which is incorrect, as shown in Fig. 4.6(b). We cannot simply apply the direction of the current adjustment vector to users' historical trajectories.

Our solution is shown in Fig. 4.6(a). During the adjustment, the server should first compute the degree of the error angle. If the time interval between the instantaneous time and the previous reposition is not too long, all of the points in the trajectories should have the same error angle. When adjusting the trajectories, we shrink or expand the length of each reported displacement and then rotate the reported trajectories with the computed error angle. The details are as follows: suppose that at time t, the server detects a false negative, and finds out that the real direct distance from the user's current location to last adjusted location is L. However, according to the reported trajectory, this direct distance is L_N. Since the systematic error

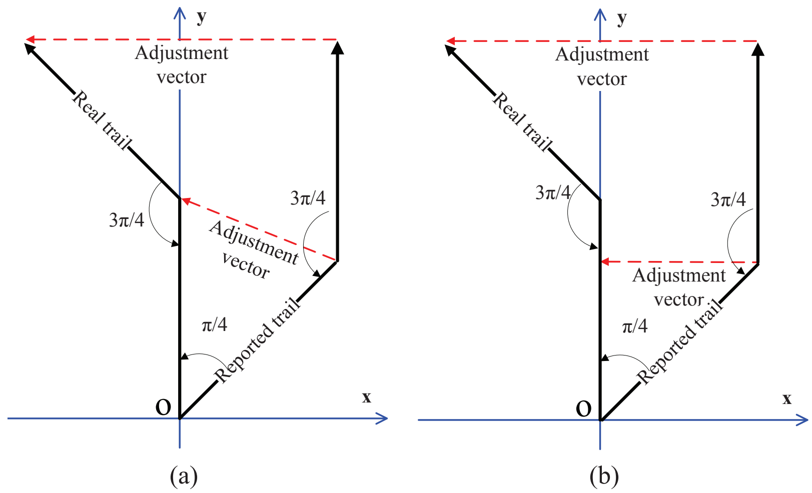


Figure 4.6: The historical error cancellation.

parameters only slightly change with time, we can regard them as fixed. Therefore, the displacement error parameter can be computed by $K = \frac{L_N}{L} - 1$. By using this computed K, we first adjust the length of each displacement, and then we rotate the whole trajectory from the last adjusted place.

4.5.5 Additional Discussion

The adjustment results of false positive and false negative should hold different weights because the false positive-based adjustment contains uncertainty; the adjustment results of false negative are much more accurate than those of false positive. With the growth of time, most of the false positive-based adjustments will eventually be re-adjusted based on newly found false negative cases. For example, t_1 is the latest time for false negative-based repositions on a pair of trajectories, and t_2 ($t_2 > t_1$) is the nearest time for false positive-based adjustment. If at t_3 ($t_3 > t_2 > t_1$) the server detects another false negative case between them, the reported trajectory from t_1 to t_3 will be adjusted. In other words, the adjustments made at t_2 are discarded. Thus, the server should also store the originally reported positions when a false positive adjustment was made. The reason that we still use the case of false positive is that, if a certain application requests the constructing map at time t_2, without applying the false positive adjustment temporarily, there will be some obviously false intersections on the visual map, which could change the relative positions of the intersections and may confuse users about the navigation results since there may be a shorter (but actually non-existing) path on the visual map but the navigation result does not go through it. As a result, we use false positive for temporary adjustments and use false negative for permanent repositioning.

4.6 Routing Application: Friend Locator

Friend locator [48] is a typical application of cooperative trajectory mapping: a server periodically collects users' trajectories and answers the routing request that helps one user to find another. The response of the routing consists of the segments from users' reported trajectories, which have spatial intersections with the others. In details, the path information consists of a series of directions and distances (such as turn right, continue for 10 m, etc.)

The implementation details of the Friend Locator application are as follows. When a user sends a routing request to the server, the server first computes the spatial intersections of the users' trajectories, which we term *inner routing nodes*. The distance between each pair of inner routing nodes is the length of the path that the user walked. We term the current position of a user as the *outer routing node*, which is linked with one, and only one, inner routing node. Every edge in the routing graph is also associated with actual walking trajectories, which consist of several displacements and turning angles. After getting the spatial intersections of the paths, the server builds a routing graph, and Dijkstra's shortest path algorithm is applied. After obtaining the shortest path, the server returns it to the requester.

The quality of this application is affected by the measurement error of the

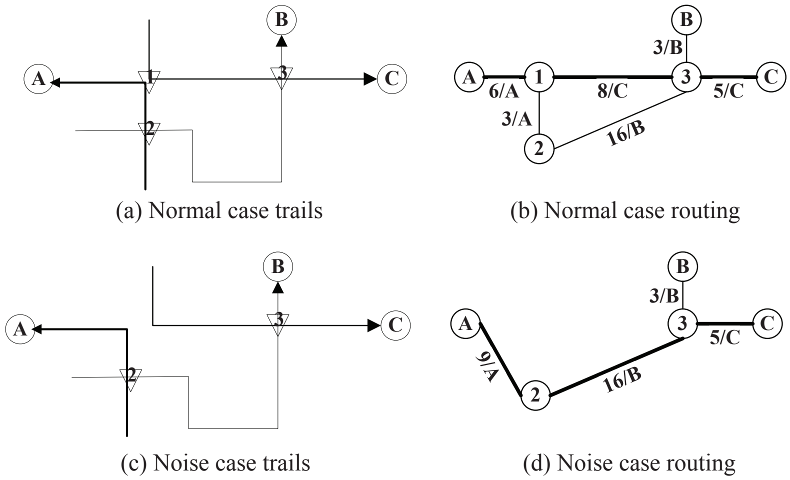


Figure 4.7: The effect of measurement error during routing.

sensor devices, which may cause both false positive intersections and false negative intersections. Fig. 4.7 shows the effect of the measurement errors on the length of the shortest routine path if one ignores the errors and directly explores all spatial intersections of the reported trajectories. In Fig. 4.7, there are three users, A, B, and C. The arrow lines represent the trajectories (i.e. walking paths) of users, and the triangles between the trajectories are the spatial intersections. Fig. 4.7 (a) is the noise-free example, and Fig. 4.7 (b) is the corresponding routing graph. The digital stands for the length of a path and the letters indicate who reported the path. Suppose that user A wants to be navigated to C. The shortest path is shown as the bold line in Fig. 4.7 (b). However, in Fig. 4.7 (c), if the trajectories of A and C are disjoint (i.e. the triangle node 1 is missing), the shortest routing becomes Fig. 4.7 (d) with its distance changed from 19 to 30. Fig. 4.7 illustrates the impacts of a false negative case, and a false positive may even cause more problems since there are wrong intersections. Due to the potential impacts of false positive and false negative intersections on the routing results, the existing navigation scheme [48] only uses the spatial intersections, where physical encounters happened, and it ignores all other spatial intersections. However, this scheme inevitably results in a long routing path. Consider that, if all of the users have the same systematic error in their sensing devices, the relative positions of their reported trajectories' intersections are still correct. Our system gradually finds out and removes the relative errors among users; for navigation, it explores not only the intersections, where physical encounters happened, but also the relatively correct spatial-intersections of the trajectories, whose relative errors are zero.

4.7 Performance Analysis and Evaluation

4.7.1 Evaluation Metric

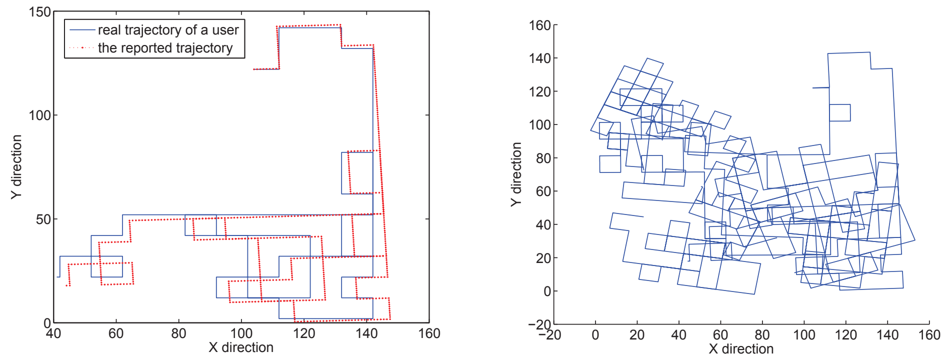
The metric we used to evaluate a navigation system is *Inaccuracy*. This is computed by applying Friend Locator, a special shortest path algorithm for trajectory map, on the adjusted trajectories. If there is an error in the routing, such as returning a non-existing path or intersection, a new routine request is created from the last existing intersection. Since the realistic walking length must be greater or equal to that of the real shortest path in the physical world, *Inaccuracy* is computed as the ratio of the extra walking distance to the length of the real shortest path within the areas that have been visited by users.

$$Inaccuracy = \left| \frac{\widetilde{d_{ij}} - d_{ij}}{d_{ij}} \right|, \tag{4.7-17}$$

where $\widetilde{d_{ij}}$ represents the total length of the real walking path, and d_{ij} is the length of the real shortest path.

4.7.2 Simulation Results

We first synthetically generate a grid map, and then, we randomly generate the noise-free moving trajectories of every user on the map. We convert the coordinates



(a) User's real & reported trajectories

(b) Reported trajectories of 10 users

Figure 4.8: An example of simulation used data.

of trajectories to sensor readings, which consist of displacement and the moving direction. The speeds of users and the systematic error parameters are randomly generated. The distribution of the parameter follows a normal distribution. The shape of noisy trajectories is shown in Fig. 4.8.

In order to guarantee that the routing request can always be responded to, we only use the trajectories which are connected. The encounter sensors' sampling times are the same as users' trajectory reporting times. Each data point in our simulation is the average result of 10 simulations. The shortest path algorithm used in our simulation is the Dijkstra algorithm. Consider that if the relative positions of users are correct, then the spatial intersections of their trajectories are correct. In order to show the importance of using the correct spatial-intersections in navigation, we compare our results with a modified version of [48]. Note that a special pruning algorithm is used in [48]. Since the pruning algorithm only affects the computing speed rather than accuracy of routing, we do not use the pruning algorithm. For ease discussion, we name the modified solution as Modified Proportional Error Cancellation (MPEC).

The first tested factor is the length of the observations. During the cooperative trajectory map construction, the relative errors among users are gradually detected and eliminated. Therefore, with the growth of time, many new correct spatial intersections will be found, and the routing results will become better and better.

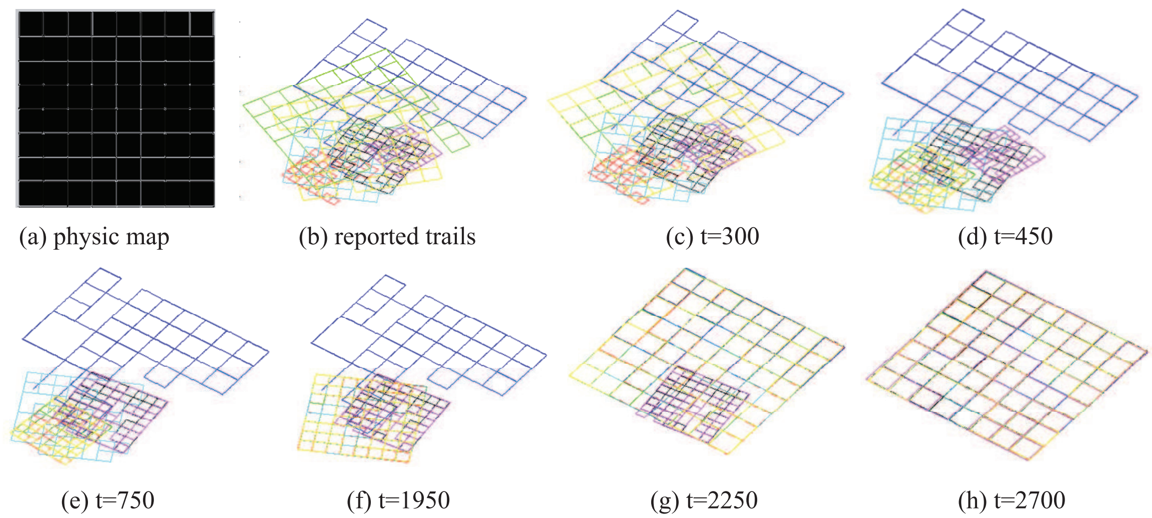
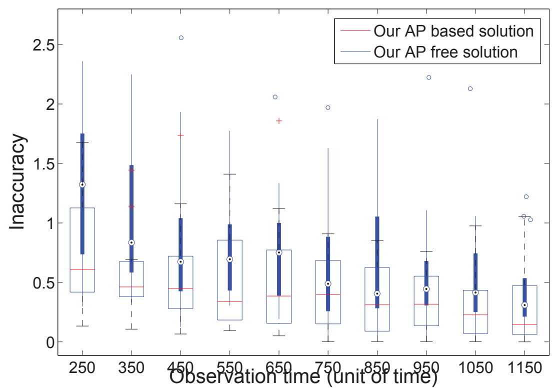


Figure 4.9: An example of the map construction process.

Fig. 4.9 shows a serial of constructing maps by using our system. Note that, in order to clearly illustrate the impacts of relative error cancellation on the final constructed map, Figs. 4.9(c)-(h) give the final maps, which are constructed by only using the encounter information until the specific times. We further compare the inaccuracy of our AP-based and AP-free solutions in Fig. 4.10. We randomly deploy 10 APs for the AP-based simulation, and we make navigation request every 100 units of time. The initial positions of the users are randomly deployed. We can see that the inaccuracy values of both solutions gradually decrease with the growth of the observation time. However, when APs are allowed, the users, who came across the same AP at different times, can be regarded as having virtual encounters, which largely speeds up the process for finding relative error parameters. Therefore, the AP-based solution has a lower inaccuracy value.

The second tested factor is the user density, as shown in Fig. 4.11. We test 10 to 16 users in the grid map. In our system, the relative error parameters are computed/updated when users encounter each other. For a close region, when there are more users, the relative error parameters can be found in a shorter time, and



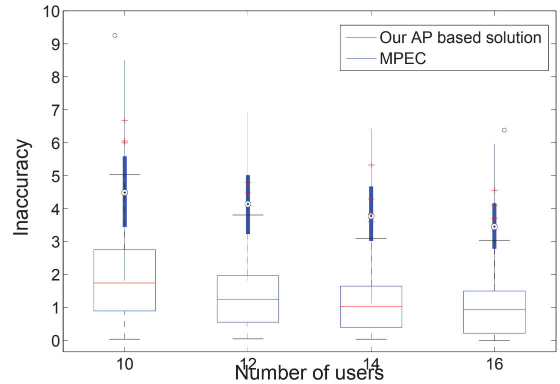


Figure 4.10: Time vs. AP/AP-free.

Figure 4.11: User num. vs. inaccuracy.

users' relative positions on the constructed map will become more accurate. Thus, with the growth of the user density, the inaccuracy decreases.

AP density is our third consideration. In the grid map, we randomly deploy APs. Intuitively, when the relative error parameters are unknown, using APs accelerates the parameter estimation process by establishing more virtual encounters. After all relative error parameters are found, if the density of an AP is large enough, the accuracy of the application will still be high even if the systematic error parameters may slightly change with time: after encountering an AP, the corresponding trajectories can be easily aligned together, and the relative error parameters can be further updated and used for corrections for a period of time. Fig. 4.12 is the simulation result. It is obviously that the accuracies of both approaches are enhanced when more APs are deployed.

Our last tested factor is the initial error parameters, which is shown in Fig. 4.13. During simulation, we first set up an initial error in sensors and let the noise slightly changes along with time. The initial errors may have significant impacts on the routing results if all trajectory intersections are directly used without error cancellation. MPEC only uses spatial intersections, where physical encounters happen, and our approach explores both the physical encounters and the spatial

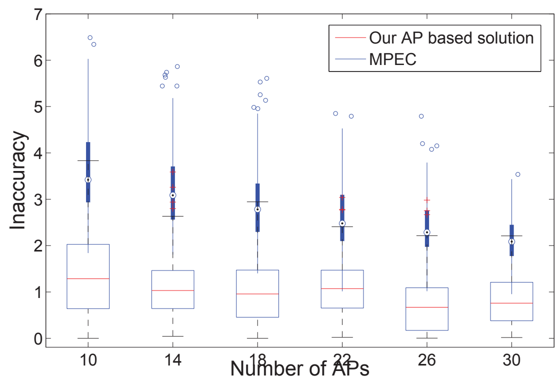


Figure 4.12: AP density vs. inaccuracy.

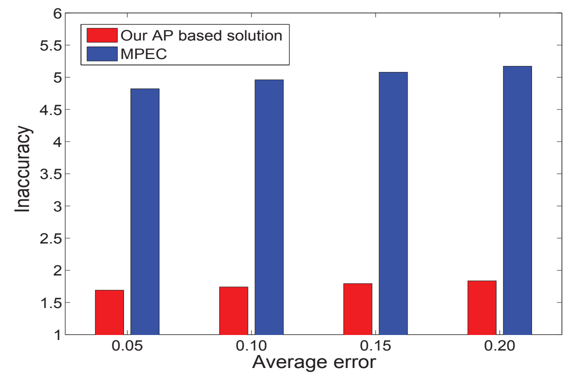


Figure 4.13: Error vs. inaccuracy.

intersections among users whose relative error parameters have been found. The errors affect the real walking distance of users, who follows the previous user's trajectory. To simulate a user's effort for searching some turning points, we give distance penalties if the relative errors between the user and the map segment's contributor are not zero. When generating the trajectories with measurement errors and when adjusting trajectories according to relative errors, the rotation and stretching operations may involve approximations. After adjusting a group of users' trajectories for eliminating their relative errors, one may find out that several trajectory intersections are very close to each other, and they are actually corresponding to the same physical intersection. A distance-based threshold is applied to determine whether these trajectory intersections should be regarded as the same one, or not. Note that this method may influence the navigation results. When the relative error parameters are unknown, once a false negative event is found, our system randomly holds one trajectory and relatively adjusts the other one. For a set of trajectories and a given threshold, the intersections may locate within a smaller-sized region after adjustment, and therefore, our system treats them as one intersection during navigation. However, if the adjustments were made according to the trajectory, who interprets the real world distance larger than the previously referenced trajectory, the intersections may locate within a bigger region. Based on the same threshold, our system may not treat them as one intersection, and the length of navigation could be longer than the previous case.

4.8 Chapter Summary

Cooperative trajectory mapping and its navigation service, friend locators, are two promising techniques, which essentially build over the existing social relations among users. In this chapter, we consider the impacts of measuring errors, especially the systematic errors, on cooperative trajectory mapping. Unlike the conventional noise model, where only random white noise is considered, we use a more general and realistic error model, where both systematic and random errors are included. Based on the new model, we propose a social encounter-based error cancellation algorithm, which focuses on eliminating the relative errors among the reported trajectories. Based on the constructed trajectory map, our system can provide a better navigation result by further exploring the relatively correct spatial intersections of users' trajectories. We use extensive simulations to validate our solution. In particular, we focus on the impact of using intersections' relatively correct positions on the performance of the shortest path routing protocol.

Here are the main contributions of this chapter:

1. We consider the impacts of systematic errors on a new social information-assisted application, called *Friend Locator*.

2. Instead of eliminating absolute errors by using physic references, we focus on removing relative errors by using social encounters, and creating a map, where intersections' relative positions are correct.

3. We provide extensive simulations to show that our approach can provide better routing results.

CHAPTER 5

PRIVACY-PRESERVED SOCIAL INFORMATION PUBLISHING FROM A CENTRALIZED SYSTEM

From Chapter 5 to Chapter 9, we study the potential security and privacy risks in social information-assisted application design. Before exploring certain social feature, one must obtain the social information from somewhere in advance. It is not a secret that, for research or commercial purpose, many sociality-related platforms publish their data set to the third party. In order to preserve the privacy of the corresponding social actors from the data set, data anonymization must be applied in advance. Privacy-preserved data publishing is becoming increasingly popular in both industry and academia. Chapters 5 and 6 respectively study the privacy-preserved sociality-data publishing in a centralized scenario and in a distributed scenario.

In this chapter, we focus on Evolving Social Subscription Networks (ESSN), which indicate social actors' participation in certain media channels, such as stars' Twitter pages, during a series of time. We introduce a new identity disclosure attack by exploring a social actor's subscribed channels' sizes and his frequency of joining/leaving the channels. Recall the video-on-demand system that we present in Chapter 2. When the system needs to publish channel subscription-related data periodically to others, an attacker may be able to re-identify certain social actors' actual identities via this attack.

For privacy protection, K-anonymity should be ensured for the whole evolving graph. However, unlike the conventional topology information, such as node degree, ESSN's data points are sparse. During the construction of anonymous groups, unpopular channel-related information is likely to be discarded. How to maximally preserve ESSN's data utility is an open problem. We propose a three-step framework to solve it: data space compression, anonymity construction, and realizable publishing. Extensive results show that our approach is effective in terms of privacy, utility, and efficacy.

5.1 Introduction

With the development of online social networks (OSN), many companies, such as Facebook and Twitter, publish their OSN data for research or marketing reasons. OSN contains many complex interactions among social actors (i.e. OSN users) [72, 21, 73], such as relations over time or across multiple media channels (i.e. Facebook fan page, an online discussion group, or a Hollywood star's Twitter). By studying the interactions, people are able to learn the information spreading pattern in real world [74], can generate personalized recommendations [75], or can even detect malicious attackers [76].

However, there is increased concern for the identity disclosure of OSN-related data publishing. OSN data is usually represented as a graph, where nodes and edges indicate social actors and their relationships. It has been shown that, on a published graph, merely replacing the real identities with some random labels cannot guarantee privacy [77, 78]: based on certain background knowledge, such as node degree [79] or local structures [80, 81], an adversary can easily reveal the physic identities of some nodes. In order to preserve privacy, K-anonymity is commonly applied such that the adversary cannot identify a node with confidence greater than $1/K$.

Most of the existing works on the privacy-preserved data publishing are based

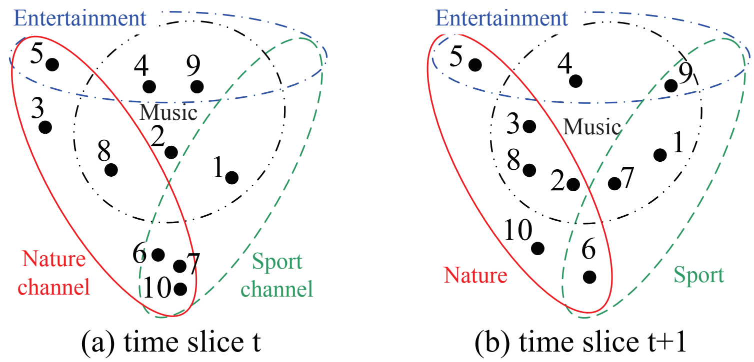


Figure 5.1: An example of evolving social subscription network. Nodes in the same circle indicate the corresponding users subscribed to the same media channel (ex. online discussion groups, or Facebook fans pages).

on a static graph (i.e. a snapshot of the OSN). In recent years, people [78, 82] have begun to consider the data publishing of evolving networks (i.e. time-varying graphs), which contain much more information than the static one. In this chapter, we focus on Evolving Social Subscription Networks (ESSN), which indicate social actors' participation in certain online media channels during a series of time. Essentially, ESSN is a hypergraph, which represents multiple relations among social actors, as shown in Fig. 5.1. When publishing the ESSN data, it is usually represented as a binary matrix, as shown in Figs. 5.2 (a)-(c).

Due to the unique structure of ESSN data, it is vulnerable to both spatial and temporal attacks. In a spatial attack, the adversary observes a single slice of the published data and tries to identify a target by exploring the size information of a target's participated media channels.

Take Fig. 5.2 as an example. Figs. 5.2(a)-(c) give social actors' channel subscriptions in three consecutive time slices. A row represents a person, a column means a channel, and 1/0 indicates whether one is subscribed or not. In Figs. 5.2(d)-(e), a channel's size is determined by the number of subscribers at time t, and frequency indicates whether the user joins/leaves a channel from t to $t+1$. $\langle + \rangle$: join new channel, $\langle - \rangle$: leave a channel, and $\langle * \rangle$: no change. In Fig. 5.2 (a), if the

t=1	C1	C2	C3	C4
V1	0	1	1	0
V2	0	0	1	0
V3	0	0	0	1
V4	1	0	1	0
V5	1	0	0	1
V6	0	1	0	1
V7	0	1	0	1
V8	0	0	1	1
V9	1	0	1	0
V10	0	1	0	1

(a) the 1st time slice

t=2	C1	C2	C3	C4
V1	0	1	1	0
V2	0	0	1	1
V3	0	0	1	1
V4	1	0	1	0
V5	1	0	0	1
V6	0	1	0	1
V7	0	1	1	0
V8	0	0	1	1
V9	1	1	1	0
V10	0	0	0	1

(b) the 2nd time slice

t=3	C1	C2	C3	C4
V1	0	1	1	0
V2	0	0	1	0
V3	0	0	1	1
V4	1	0	1	0
V5	1	0	0	1
V6	0	1	0	1
V7	0	1	0	1
V8	0	0	0	1
V9	1	0	0	1
V10	0	1	1	0

(c) the 3rd time slice

	Channel Size	Frequency
V1	⟨4,5⟩,⟨4,6⟩,⟨4,5⟩	⟨*⟩,⟨*⟩
V2	⟨5⟩,⟨6⟩,⟨5⟩	⟨+⟩,⟨-⟩
V3	⟨6⟩,⟨6,6⟩,⟨5,6⟩	⟨+⟩,⟨*⟩
V4	⟨3,5⟩,⟨3,6⟩,⟨3,5⟩	⟨*⟩,⟨*⟩
V5	⟨3,6⟩,⟨3,6⟩,⟨3,6⟩	⟨*⟩,⟨*⟩
V6	⟨4,6⟩,⟨4,6⟩,⟨4,6⟩	⟨*⟩,⟨*⟩
V7	⟨4,6⟩,⟨4,6⟩,⟨4,6⟩	⟨+,-⟩,⟨+,-⟩
V8	⟨5,6⟩,⟨6,6⟩,⟨6⟩	⟨*⟩,⟨-⟩
V9	⟨3,5⟩,⟨3,4,6⟩,⟨3,6⟩	⟨+⟩,⟨+,-⟩
V10	⟨4,6⟩,⟨6⟩,⟨4,5⟩	⟨-⟩,⟨+,-⟩

(d) spatial temporal information

Size	C1	C2	C3	C4
t=1	3	4	5	6
t=2	3	4	6	6
t=3	3	4	5	6

(e) channel size sequence

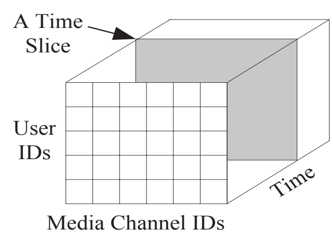


Figure 5.2: An example of attacks on ESSN.

attacker knows that Bob subscribes to two channels with sizes 5 and 6, then he can infer that node v_8 is Bob. In a temporal attack, the adversary is able to use the target's frequencies of joining or leaving the channels to identify him. For instance, in Fig. 5.2 (a)-(c), although the sizes of v_6 and v_7's subscribed channels are the same in all time slices, once the adversary knows that Alice has never changed her subscriptions during that period, he can uniquely locate the corresponding node of Alice, which is v_6. In addition, this spatial-temporal background knowledge is easy to obtain from most online social network platforms.

In order to defend the new identity disclosure attack, on a published ESSN graph, we must ensure that any potential target's temporal sequence regarding his subscribed channels' sizes and joining/leaving frequency is indistinguishable from that of at least $K-1$ other nodes. In the meanwhile, it must retain as much of the original graph's information as possible. Compared to the static graph-related anonymization, the privacy-preserved ESSN data publishing is more of a challenge in three aspects. First, unlike the conventional topology information, such as node degree, ESSN's data points are much more sparse. For instance, if a graph contains 25 channels' subscription information, the probability for any two nodes having the same subscriptions at a time is $1/2^{25}$. How to partition the sparse data points into several anonymous groups directly affects the final published graph's utility. Second, the members of an anonymous group should have the same final joining/leaving frequency and subscribed channel size, which is usually determined by the majority. However, it is likely to cause the unpopular channel-related data being discarded. Third, changing a node's subscription will cause the modification of other anonymous groups' channel size and frequency; there are issues of consistency and realizability. How to efficiently and maximally preserve ESSN's data utility under the new attack is an open problem.

In this chapter, we formalize the spatial and temporal information-based ESSN anonymization problem, and we propose an effective three-step framework to solve it. Due to the high dynamics of the channel size and frequency-related information, one has to combine several different distance metrics for grouping, which is very time-consuming. We notice that the inappropriate grouping usually happens when nodes subscribe to different numbers of channels: the closeness is submerged by the matching distance of the missing subscriptions. For reducing the happening of these two situations, before creating anonymous groups, we first compress and coarsely partition the data space according to the similarity of channels' temporal information. In order to preserve the utility of unpopular channels, we introduce a new concept called equivalent channels: if a set of channels always have the same size and joining/leaving frequency, then the adversary cannot discriminate their subscribers from each other, purely based on these channels' spatial-temporal information. Extensive experiment results show the effectiveness of our approach in terms of privacy, utility, and efficacy.

5.2 Literature Review

This work focuses on publishing an anonymized social subscription network such that, under STA, each social actor's privacy is preserved. Unlike the conventional papers, we use a more challenging system model, where the original data is a time-varying graph, consisting of T discrete temporal slices. To the best of our knowledge, only a few papers [78, 82] have considered the scenarios of continuously privacy-preserved data publishing. Moreover, these papers only consider very simple topology information, such as node degrees, and their problem only needs to anonymize one type of information, while we have to make both NRS and NFS indistinguishable among a set of nodes. The subscribed channels' size-related attack was first introduced by paper [83]. However, their attack model is only based on a static graph, and they used a channel-unspecified distance to construct anonymous groups. However, once the time dimension is disclosed, the adversary can easily track the membership of any anonymous group, which can shrink the number of indistinguishable nodes, or the attacker even uses the NFS to identify a target.

There are two types of re-identification attacks: active or passive attacks [77]. In an active attack, an adversary intentionally embeds a special graph into a social network before publishing and uses the knowledge about this graph to infer the identity of nodes from the published network. In a passive attack, the adversary possesses some knowledge about a target node and re-identifies the target by searching through the published data with the knowledge. Essentially, both types of attacks are based on graph matching. On a published social graph, if the adversary is able to uniquely pinpoint a node satisfying the knowledge about the target, the privacy of the social actor behind the node will be disclosed.

To defend the re-identification attacks, most papers [84, 85, 78, 83] apply the k-anonymity model: on an anonymized social graph, each node should have at least $k-1$ partners such that no one can distinguish them based on the associated structural information. To achieve k-anonymity, paper [80] builds the anonymous groups by creating automorphism graphs; work [86] groups similar nodes and builds isomorphism among them; paper [87] partitions nodes into different groups and only discloses the information at a group-level; article [88] injects edges with probabilities of existence in a published social graph. Among all these schemes, isomorphism-based graph anonymization is the main trend: Paper [79] creates isomorphic node degrees, and works [81, 89] build isomorphisms on a local 1-hop graph. In this chapter, we build anonymous groups by creating isomorphic NRS and NFS.

5.3 System Model and Problem Formulation

5.3.1 System Model

We consider the privacy-preserved data publishing over a time-varying social subscription network. The system contains m channels (i.e. $C = \{c_1, c_2, \ldots, c_m\}$). In reality, the channels could be a star's Twitter, a company's Facebook page, or an online discussion group. Based on users' social interests, he may follow the account, like the page, or join the group. For the ease of description, we generally call these actions 'channel subscriptions,' and users can add or cancel the subscriptions. At time slice t, the network is represented as a hypergraph $G_t = (V, E_t)$ over a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$. Let $S(v,t)$ denote the set of channels that user v subscribed to by time t. A hyperedge $e_k^t \in E_t$ is defined as $e_k^t = \{v_i, v_j, \ldots | \forall v_i, v_j \in V, c_k \in S(v_i,t) \bigcap S(v_j,t)\}$. For a given time, the mapping between a hyperedge and the corresponding channel is bijection. A data owner has T snapshots of the network, and these snapshots form the evolving graph to be published, $\mathbb{G} = \{G_1, G_2, \ldots, G_T\}$.

We use a $n \times m \times T$ boolean matrix to store the whole evolving graph $\mathbb{G}$, as illustrated by Figs. 5.2 (a)-(c).

5.3.2 Attack Model

Privacy-preserved subscription data publishing was first proposed by paper [83]. However, their model only considers the privacy on a static graph. Once a published subscription graph contains more than one temporal slice, it is vulnerable to spatial and temporal attacks.

Definition 5.1. Edge Rank Let $\gamma(\cdot)$ be the edge rank, which is defined as the number of users subscribed to a channel at time t, $\gamma(e_k^t) = |e_k^t|$, where $|\cdot|$ denotes the cardinality of a set. $\gamma(e_k^t) = 0$ if v_i has not subscribed to c_k at time t.

Definition 5.2. Node Rank The rank of v_i at time t consists of all related edge ranks, $R_{it} = \langle \gamma(e_k^t) \rangle$, for $\forall c_k \in S(v_i, t)$.

For example, in Fig. 5.2 (a), the vertices v_1 subscribes to c_2 and c_3. The channel c_2 currently has 4 subscribers, and c_3 has 5, and therefore, the rank of v_1 is $\langle 4, 5 \rangle$. Similarly, for v_3 in Fig. 5.2 (b), since both of its subscribed channels have 6 subscribers, the node rank is $\langle 6, 6 \rangle$.

Definition 5.3. Spatial Attack Given G_t, a spatial attack is successful if the attacker can uniquely identify v_i from G_t based on the background knowledge about a target's rank R_{it}.

Take Fig. 5.2 (a) as an example. If an adversary knows that Alice only participates in a channel having 5 subscribers at $t = 1$, then he is able to infer that v_3 is Alice. Clearly, the success rate of a spatial attack is related with the uniqueness of a user's subscribed channels' ranks. For the users who subscribe to some unpopular channels, it is likely that the attacker can find them. However, publishing an evolving graph gives more information, even for the users who only follow a few popular groups.

For instance, in Fig. 5.2, v_4 and v_9 have the same node rank at $t = 1$, but once the attacker knows a target's rank sequence in the first two slices, $\langle 3, 5\rangle, \langle 3, 6\rangle$, he also can infer that the target's corresponding node is v_4.

Definition 5.4. Node Rank Sequence (NRS) v_i's temporal rank sequence is represented as $R_{i\cdot} = [R_{i1}, R_{i2}, \ldots, R_{iT}]$.

Even if a set of nodes have the same NRS, the adversary may still be able to identify some of them by exploring the frequency of joining/leaving channels. In Figs. 5.2 (a)-(c), v_6 and v_7 always have the same rank, but, if the adversary knows that Charles changed subscriptions from t_1 to t_2, and t_2 to t_3, then based on the changing pattern (Fig. 5.2 (d)), v_7 is likely to be Charles. Here, we only consider the information about whether a user joins or leaves a channel. The general definitions of node frequency sequence and the temporal attack are as follows.

Definition 5.5. Node Frequency Sequence (NFS) Node v_i's channel modification frequency from t to $t+1$ is represented as $F_{it} = \langle |S(v, t+1) \backslash S(v, t)|, |S(v, t) \backslash S(v, t+1)|\rangle$. v_i's frequency sequence is $F_{i\cdot} = [F_{i1}, \ldots, F_{i(T-1)}]$.

For example, in Fig. 5.2, $F_{1\cdot} = [\langle 0, 0\rangle, \langle 0, 0\rangle]$ since v_1 did not change the subscriptions, and $F_{9\cdot} = [\langle 1, 0\rangle, \langle 1, 2\rangle]$ since v_9 added one channel subscription from t_1 to t_2, canceled two subscriptions and added a new one from t_2 to t_3.

Definition 5.6. Temporal Attack For a given evolving graph $\mathbb{G}$, if an attacker can uniquely find node v_i based on his background knowledge $F_{i\cdot}$, then the privacy of v_i is disclosed.

In general, we call the attack using any combination of NRS and NFS *Spatial-Temporal Attack* (STA). In reality, launching a STA is easy. For most social networks, such as Facebook and RenRen, the real-time membership of a channel and its size are shown to the public. As long as the attacker knows the data collecting time, he can always launch the STA. When releasing a data set, the data owner usually publishes the data collecting time such that people can verify them. Admittedly, it may be hard for an attacker to obtain the exact channel size at a given time; however, he is surely able to find out an approximated size value. For instance, the subscribed channel sizes that the data owner collected are $R_{it} = \langle 125, 352, 1330 \rangle$, and the attacker may get $\widetilde{R_{it}} = \langle 143, 358, 1342 \rangle$. In this case, during the attack, the attacker first defines several magnitude levels, such as l_0: $(0, 99]$, l_1: $(99, 199] \ldots$, and then, he converts the channel sizes on both the attacking graph and background knowledge to the magnitude levels, and uses them to construct the target user's vertices rank sequence. Then, both R_{it} and $\widetilde{R_{it}}$ become $\langle l_1, l_3, l_{13} \rangle$. Hence, it is necessary to consider the STA-related privacy leakage in ESSN.

5.3.3 Problem Definition

From a privacy aspect, our goal is to partition the nodes of an evolving graph into disjoint K-anonymous groups, such that the members of each group are indistinguishable, based their NRS and NFS. Note that, in our model, two nodes with the same NRS (or NFS) does not indicate that the corresponding users subscribe to the same channel. For instance, Alice subscribes to a perfume channel, whose size changes as $2 \rightarrow 4 \rightarrow 8$. Bob subscribes to a beer discussion page, and the page's size also changes as $2 \rightarrow 4 \rightarrow 8$. Bob potentially could join the same anonymous group as Alice.

From a utility aspect, we want to maximally preserve the original graph's utility, which corresponds to construct the anonymous groups by using a minimum number of subscription modifications. There are two types of modifications: adding a subscription (i.e. change $0 \rightarrow 1$ in a subscription table, like Fig. 5.2(a)) and canceling a subscription (i.e. change $1 \rightarrow 0$ in the table), and we do not differentiate them.

Assume that an anonymized graph $\mathbb{G}'$ contains a total of l anonymous groups.

Let h be a $n \times l$ boolean matrix: if $h_{ik} = 1$, then v_i is in the anonymous group g_k. Let $S'(v_i, t)$, $R'_{i\cdot}$, and $F'_{i\cdot}$ denote v_i's published subscriptions, NRS, and NFS on $\mathbb{G}'$, respectively. For saving more utility, we allow to use at most αn dummy nodes, where $\alpha \in (0, 1]$. The evolving graph's anonymization problem is defined as follows:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{l} h_{ij} \times \sum_{t=1}^{T} dist[S(v,t), S'(v,t)] \\
\text{s.t.} \quad & R'_{i\cdot} = R'_{j\cdot}, F'_{i\cdot} = F'_{j\cdot}, \qquad \text{for } \forall i, j, h_{ik} = h_{jk} = 1 \\
& \textstyle\sum_{j=1}^{l} h_{ij} = 1, \qquad \text{for } \forall i \in [1, n] \\
& \textstyle\sum_{j=1}^{l} u_{kj} = \{0, 1\}, \qquad \text{for } \forall k \in [1, \alpha n] \\
& \textstyle\sum_{i=1}^{n} h_{ij} + \sum_{k=1}^{\alpha n} u_{kj} \geq K, \qquad \text{for } \forall j \in [1, l]
\end{aligned}$$

Let $|\cdot|$ be the cardinality of a set, then $dist[S(v,t), S'(v,t)] = |S(v,t) \backslash S'(v,t)| + |S'(v,t) \backslash S(v,t)|$. The constraints guarantee that the members of an anonymous group have the same NFS and NRS, each node belongs to one and only one anonymous group (dummy node joins at most one group), and the size of any group is not less than a pre-defined K. The problem is NP-complete. The basic proof idea is to reduce it to the NP-complete problem, planar exact cover by 3 sets [90].

5.4 Overview

The goal of this chapter is to use a minimal number of 0/1-changes to create a K-anonymized evolving graph such that every node is indistinguishable from at least another $K-1$ nodes based on their NRS and NFS. It is an NP-complete problem, and we propose greedy algorithms with high efficacy to approximate it. For the privacy requirements, we notice that the information provided by NRS and NFS are partially overlapped: for any two consecutive time slices, unless a social actor joins and leaves the same number of channels with the same sizes, all other joining/leaving events can be captured by NRS. In this chapter, we approximate the optimal ESSN data anonymization by first constructing anonymous groups according to the similarity of NRS and then enforcing a uniform NFS within each group.

The distance metric measuring the similarity of nodes directly influences the final constructed anonymous group. But, the distance calculation between a pair of social actors must be conducted in the same space. The unique data structure of NRS put us in a dilemma: both the channel specified distance metric and unspecified one have defects. In a channel specified distance metric, each media channel is mapped to a unique data dimension, and the value at a dimension is determined by the corresponding hyperedge's rank. However, unless users subscribe to the same channels, which is rare, the metric will inevitably make the NRS data points sparse (grouping becomes meaningless when data points no longer cluster together).

Paper [83] adopts a channel-unspecified metric: for a node rank, it sorts rank values in descending order, and fills the missing dimension with zeros such that all nodes have the same number of dimensions. The distance is calculated as the norm-2 distance. For instance, assume that users subscribe to at most 3 channels. The data point of v_i's node rank, $R_{it} = \langle 38, 60 \rangle$, is represented as $\langle 60, 38, 0 \rangle$. There are another two users v_j and v_k, with $R_{jt} = \langle 48, 30, 21 \rangle$ and $R_{kt} = \langle 60, 40, 37 \rangle$. The distances between them are computed as $dist(v_i, v_j) = \sqrt{(60-48)^2 + (38-30)^2 + (0-21)^2} = \sqrt{649}$ and $dist(v_j, v_k) = \sqrt{500}$. Compared to v_i, it seems v_j is closer to v_k. However, in reality, putting v_i and v_j in the same anonymous group costs much less effort than putting v_j and v_k together since the former case only needs to add one more channel subscription, while the later one needs membership modifications in three channels. The essential problem is that the matching distance caused by the missing dimensions fully conceals the differences of the other dimensions.

In order to save more utility, we design a mixed strategy for the distance calculation. At a higher level, we use a channel-group-specified distance metric to compress and partition the data space into several regions, and at a lower level, anonymous groups are constructed based a channel-unspecified metric. When social actors subscribe to different numbers of channels, our approach forces data points only to compare with some suitable points, which are partially similar to each other, instead of the nodes, who have same dimension but totally different rank values. After anonymous groups are formed, each group will find a group-based NRS. However, the modification of a user's subscription will also change the rank of his subscribed channels. There are consistency and realizability issues among the NRS values of different groups. We design an algorithm to solve it and enforce the NFS-indistinguishability at the same time. All of the above approaches build up a three-step framework: space compression, anonymity construction, and realizable publishing.

5.5 Space Compression

In order to avoid inappropriate grouping and enhance the efficacy of the anonymization algorithm, we compress the original sparse data space into a smaller one, such that users, who subscribe to similar-sized channels, are close to each other in the new space. Based on the clustering, space is further partitioned into subspaces, and anonymous groups are constructed within each subspace.

We first define the channels' similarity as the norm-1 distance between their edge ranks at each temporal slice.

$$dist(c_i, c_j) = \sum_{t=1}^{T} \sqrt{[\gamma(e_i^t) - \gamma(e_j^t)]^2}$$

Algorithm 12 The construction of the mapping function $\mathcal{M}$

1: Create same-position groups: $g_i \leftarrow \{c_i\}, \forall i \in [1, m]$
2: **while** $|\{g_i | g_i \neq \phi\}| > \lceil \beta l \rceil$ **do**
3:     **for** Every pair of non-empty groups g_i and g_j **do**
4:         Compute the group merging cost: $\frac{\sum_{c,c' \in g_i \bigcup g_j} dist(c,c')}{|g_i| + |g_j|}$
5:     Find the group pairs g'_i, g'_j with the smallest merging cost
6:     $g'_i \leftarrow g'_i \bigcup g'_j$ and $g'_j \leftarrow \phi$
7: Based on the non-empty position groups $\{g\}$, create function $\mathcal{M}$

Let β be a system-defined parameter, which controls the total number of dimensions in the compressed space. In the new space, a node rank at time t is represented by a vector $P_i[t] = (P_i[t][1], P_i[t][2], \ldots, P_i[t][\beta])$, and node v_i's NRS is stored in a $T \times \beta$ matrix, $P_i[\cdot][\cdot]$. Let $\mathcal{M}(c)$ denote a mapping function from a channel to a dimension d in the new space, $1 \leq d \leq \beta$, and $\mathcal{M}^{-1}(d)$ indicate a set of channels mapped to d. The function $\mathcal{M}(\cdot)$ is constructed such that the overall mapping costs to the β dimensions are minimized as follows:

$$\min \sum_{d=1}^{\beta} \sum_{c_i, c_j \in \mathcal{M}^{-1}(d)} dist(c_i, c_j)$$

We propose a hierarchical clustering scheme to create the mapping function $\mathcal{M}(\cdot)$, as shown in Algorithm 12. In line 1, each channel is assigned with a unique group. When the system contains more than β non-empty groups, Algorithm 12 repeatedly computes the average costs for merging them (line 4), and always combines a pair of groups with the smallest cost (lines 4-5). Algorithm 12 stops when it contains exactly β non-empty groups, and $\mathcal{M}(\cdot)$ is constructed based on the remaining groups: if $c, c' \in g$, then $\mathcal{M}(c) = \mathcal{M}(c')$; if $c \in g$, $c' \in g'$ and $g \neq g'$, we have $\mathcal{M}(c) \neq \mathcal{M}(c')$.

Based on $\mathcal{M}$, the coordinates (i.e. P_i) of v_i in the compressed space are determined as follows: if $c_j \in S(v_i, t)$, then $P_i[t][\mathcal{M}(c_j)] = |\gamma(e_j^t)|$. However, it is inevitable to have collisions: if $\exists v_i, t$ such that $c, c' \in S(v_i, t)$ and $\mathcal{M}(c) = \mathcal{M}(c')$, then the situation is called "collision". In order to solve it, channels are assigned with different priorities, which are based on their overall number of subscribers (i.e. $\sum_{t=1}^{T} |\gamma(e_i^t)|$). When collision happens, the corresponding dimension is taken by the channel with a higher priority. The advantage of this approach is to keep the data points of large channels together.

The compressed space is partitioned by the K-means algorithm. The distance between any two social actors is calculated as the sum of absolute differences between their coordinates: $dist(P_i, P_j) = \sum_{t=1}^{T} \sum_{k=1}^{\beta} \sqrt{(P_i[t][k] - P_j[t][k])^2}$. Note that, due to the collision, directly creating anonymous groups in the new space will inevitably cause unnecessary utility loss, especially for a small anonymous group size K. Here, we are interested in grouping the nodes who subscribe to the channels with similar ranks such that, during the later refine phase, nodes with very different subscriptions will not join the same anonymous group.

5.6 Anonymous Group Construction

Within each partitioned subspace, nodes will be further divided into different anonymous groups according to the similarity of their *original* NRS values. Note that, although this session purely uses NRS to construct anonymous groups, we still cannot ignore the existence of NFS during the algorithm design. For the spatial attack on a single time-slice graph, as long as the nodes from an anonymous group have the same node rank, that group is privacy-preserved. For example, in Fig. 5.2(b), although v_4 and v_5 subscribe to different channels, they are privacy-preserved (i.e. 2-anonymous) since the attacker cannot discriminate them based on the rank value. So, from a privacy aspect, since there is no difference for a user to subscribe to either c_3 or c_4, these two channels are *equivalent* at that time. However, in ESSN, due to the existence of NRS and NFS information, such an equivalency should be built based on users' personal temporal-subscription sequences.

Let $R_{it}(c_k)$ denote the edge rank of v_i's subscribed channel c_k at time t. The matching distance between a pair of users' any two channels' rank sequences are defined as follows: $dist\left[R_{i\cdot}(c), R_{j\cdot}(c')\right] = \sum_{t=1}^{T} \sqrt{\left[R_{it}(c) - R_{jt}(c')\right]^2}$. Note that, even if both v_i and v_j subscribe to c_k, $R_{i\cdot}(c_k)$ may be unequal to $R_{j\cdot}(c_k)$ since v_i and v_j could subscribe to the channel at different times. If v_i has not subscribed to channel c at time t, then $R_{it}(c) = 0$. For instance, in Fig. 5.2, both v_9 and v_{10} have subscribed to c_3, but $R_{9\cdot}(c_3) = [5, 6, 0]$ and $R_{10\cdot}(c_3) = [0, 0, 5]$. Since users may subscribe to different numbers of channels, their NRS values could have different dimensions. How to define node similarity is the key of constructing anonymous groups.

In this chapter, we define a channel-semi-specific similarity metric. For any two data points from the same subspace, we use the minimum number of changes for transforming one point's NRS to another as their similarity. Let $S(v_i)$ denote the set of channels to which v_i has subscribed, $S(v_i) = \bigcup_{t=1}^{T} S(v_i, t)$. The similarity calculation of v_i and v_j consists two parts: For the channels subscribed by both of them ($c_k \in S(v_i) \bigcap S(v_j)$), their distance is calculated as $dist\left[R_{i\cdot}(c_k), R_{j\cdot}(c_k)\right]$; For the channels subscribed by either v_i or v_j, $c_k \in (S(v_i) \backslash S(v_j)) \bigcup (S(v_j) \backslash S(v_i))$, we find one unchecked channel subscribed by the other user, which has the smallest distance with c_k, and compute their distance. For example, suppose that v_i follows channel c_1 from t to $t+2$ (the size changes as $3 \to 8 \to 7$) and channel c_2 from $t+1$ to $t+2$ (the size is $7 \to 18$); $R_{i\cdot}(c_1) = [3, 8, 7]$, and $R_{i\cdot}(c_2) = [0, 7, 18]$. v_j only follows c_3, and $R_{j\cdot}(c_3) = [9, 7, 17]$. The matching distance between v_i and v_j is $(3 + 8 + 7) + (|0 - 9| + |7 - 7| + |18 - 17|) = 28$ when converting v_i's c_2-related NRS values to v_j's NRS.

For showing the effectiveness of the channel-semi-specific metric, we also propose a channel-specific one. Let $\rho_{ij}(c_k)$ denote the subscription similarity of v_i and v_j for channel c_k, and let ρ_{ij} represent their overall subscription similarity. Considering that the channel size is an important factor during the construction of anonymous group,

ρ_{ij} is defined as: $\rho_{ij} = \frac{1}{m}\sum_{k=1}^{m}\rho_{ij}(c_k)$, where $\rho_{ij}(c_k) = \frac{\sum_{t=1}^{T}|\{e_k^t|v_i,v_j\in e_k^t \text{ or } v_i,v_j\notin e_k^t\}|}{\sum_{t=1}^{T}|e_k^t|}$. Note that ρ_{ij} purely reflects users' similarity for subscribing to the exactly same channels. Whether a pair of nodes belongs to the same anonymous group depends on their similarity. In Chapter 5.8, we compare these two metrics and show that the channel-semi-specific one saves more utility. However, here, we collectively call the results of the metrics 'matching distance', and use d_{ij} to represent it.

Within each subspace created in Chapter 5.7, we use a bottom-up hierarchical strategy to build the anonymous groups, as shown in Algorithm 13. The algorithm first treats each node as a unique anonymous group (line 2) and gradually merges some groups into a larger one. For any two non-empty groups, whose size is smaller than K, it computes the average cost for merging them (lines 4-6). Once the size of a resulting group is no less than the required anonymity level K, the algorithm will not modify its membership anymore. Let n_g be the number of non-empty groups, whose size is smaller than K, and let n_v be the total number of nodes from these groups. If we want each of these groups to form a K-anonymous group individually, we need at least $Kn_g - n_v$ dummy nodes. When the smallest merging cost is greater than a pre-defined threshold and the system contains a large enough number of dummy nodes (lines 7-8), the algorithm stops merging and adds dummy nodes to each of the remaining groups, whose size is smaller than K (line 11). All of the non-empty groups are the constructed anonymous groups.

On the final graph $\mathbb{G}'$, all the members of an anonymous group must have the same NRS (also NFS). So, we need to select a Group-level NRS (G-NRS) for each constructed group. Later, the original ESSN graph will be modified according to it; an inappropriate G-NRS will cause more utility loss.

During the construction of G-NRS, two issues should be considered. First, when does an anonymous group join/leave a channel? Consider that, when adds/removes

Algorithm 13 The construction of anonymous groups

Compute the distance d_{ij} between any pair of nodes
Create anonymous groups: $g_i \leftarrow \{v_i\}, \forall i \in [1, n]$
while $\exists g, 0 < |g| < K$ **do**
  for Any groups g and g' satisfying $|g|, |g'| \in [1, K)$ **do**
    Compute grouping cost: $c_g \leftarrow \sum_{v_i \in g, v_j \in g'} d_{ij}/(|g| + |g'|)$
  Find the groups g_i, g_j with the smallest cost
  $n_v \leftarrow \sum_{0<|g|<k}(K - |g|)$, $n_g \leftarrow |\{g|0 < |g| < k\}|$
  if $c_g \le c_g^* \parallel n_v + \alpha n < K \cdot n_g$ **then**
    $g_i \leftarrow g_i \bigcup g_j$ and $g_j \leftarrow \phi$
  else
    Add dummy nodes to the remaining groups

nodes to/from a channel, the node ranks of the channel's other subscribers will also be changed. Modification costs are related to, not only the number of nodes that have been added/removed, but also the channel's size. Given an anonymous group g and time t, the cost for letting the existing subscribers to leave a channel c_k is about $\gamma(e_k^t)|\{v|v \in g, v \in e_k^t\}|$, while the cost for letting non-subscribers to join c_k is $\gamma(e_k^t)|\{v|v \in g, v \notin e_k^t\}|$. At any time, a group's joining/leaving decision is determined by choosing the action with a less cost (function $cost_I$ in Algorithm 14).

The second issue is whether to build equivalent channels. Consider that some channels may not be popular. In order to save more data utility, we convert certain channels, who have similar size-changing patterns, into equivalent channels such that they always have the same size; the subscribers of these channels can be put into the same anonymous group, even if they subscribe to different ones. When c_k and c_k' are used as equivalent channels, the minimal cost for an anonymous group g to form a G-NRS is given by function $cost_E$ in Algorithm 14: For each time t, if the total number of group members subscribed to either c_k or c_k' is less than half of the group size, these subscribers will be removed from the channels, the cost of which is given by line 19; otherwise, the whole group will use the larger channel's rank, $\max(\gamma(e_k^t), \gamma(e_{k'}^t))$. The cost of this operation comes from three parts: (1) joining costs for the group members, who are not subscribing to either c_k or c_k'; (2) the costs for letting the members

Algorithm 14 Equivalent channels' construction

if $\sum_g(cost_I(c,g) + cost_I(c',g) - cost_E(c,c',g)) > 0$ **then**
  Make c_k and c'_k NRS equivalent

Function $cost_I(c_k, g)$:
$Cost_I \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
  if $|\{v|v \in g, v \in e_k^t\}| \geq |g|/2$ **then**
    $Cost_I \leftarrow Cost_I + \gamma(e_k^t) \cdot |\{v|v \in g, v \notin e_k^t\}|$
  else
    $Cost_I \leftarrow Cost_I + \gamma(e_k^t) \cdot |\{v|v \in g, v \in e_k^t\}|$

Function $cost_E(c_k, c'_k, g)$:
$Cost_E \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
  if $|\{v|v \in g, v \in e_k^t \parallel v \in e_{k'}^t\}| \geq |g|/2$ **then**
    if $\gamma(e_k^t) \geq \gamma(e_{k'}^t)$ **then**
      $Cost_E \leftarrow Cost_E + (\gamma(e_k^t) - \gamma(e_{k'}^t))|\{v \notin e_k^t, v \in e_{k'}^t\}| + \gamma(e_k^t)|\{v \notin e_k^t, e_{k'}^t\}| + \gamma(e_{k'}^t)|\{v \in e_k^t, e_{k'}^t\}|$
    else
      $Cost_E \leftarrow Cost_E + (\gamma(e_{k'}^t) - \gamma(e_k^t))|\{v|v \notin e_{k'}^t, v \in e_k^t\}| + \gamma(e_{k'}^t)|\{v|v \notin e_k^t, e_{k'}^t\}| + \gamma(e_k^t)|\{v|v \in e_k^t, e_{k'}^t\}|$
  else
    $Cost_E \leftarrow Cost_E + \gamma(e_k^t)|\{v|v \in e_k^t\}| + \gamma(e_{k'}^t)|\{v \in e_{k'}^t\}|$

subscribed to both channels leave the smaller one; and (3) the cost for increasing the smaller channel's size to $\max(\gamma(e_k^t), \gamma(e_{k'}^t))$. Given two channels, Algorithm 14 compares the overall costs for keeping them independent ($cost_I$) in all anonymous groups and the costs for using them as equivalent channels ($cost_E$). If enforcing equivalency can save more utility, Algorithm 14 always keeps the channels having the same rank. Our system first builds several pairs of equivalent-channels, and then determines the final G-NRS (with a minimal cost) for each anonymous group.

5.7 Realizability and NFS Indistinguishability

A channel's subscribers may belong to different anonymous groups. Although each constructed anonymous group has its own G-NRS, which has the minimal modification costs to construct the corresponding graph, the G-NRS values from different groups may conflict. For example, $v_i, v_j \in e_k^t$ but $v_i \in g, v_j \in g'$; because

G-NRS is independently created by each group, it may occur that the corresponding G-NRS of g is 10, while that of g' is 15; clearly, inconsistency happens, since hyperedge e_k^t can only have one rank at a time slice. Moreover, suppose there is a channel with 5 subscribers but all of them require the corresponding rank value as 4. Since a channel's rank is determined by the number of subscribers, the required value is not realizable. Therefore, different anonymous groups' G-NRS should be adjusted. Moreover, the NFS indistinguishability should be enforced into every group.

The indistinguishability of NRS and NFS in an anonymous group can be achieved by either letting all its members always join/leave the same channels together, or letting the members simultaneously subscribe/unsubscribe to *one* of the equivalent channels that always have the same rank. Channels can be categorized as independent (regular) channels and equivalent channels, which have been established in the previous section. The problem of constructing realizable G-NRS and enforcing NFS indistinguishability will be solved by different approaches according to the types of related channels.

Let $S(g,t)$ denote a set of channels that the G-NRS of anonymous group g contains these channels-related ranks at time slice t. For an independent channel c_k, if $c_k \in S(g,t)$, the original tth ESSN subscription table will be modified such that the corresponding members' values under the c_k column are 1s (i.e. subscribed); similarly, if $c_k \notin S(g,t)$, the values are 0s (i.e. unsubscribed). In order to guarantee that different groups' c_k-related G-NRS are realizable, the corresponding G-NRS values are set to $\sum_{\forall g} |\{v_i | v_i \in g, c_k \in S(g,t)\}|$.

Equivalency is a transitive property, and therefore, the constructed equivalent channels build up several equivalent-channel groups (e-group for short). The channels of an e-group may be subscribed by nodes from different anonymous groups, which join/leave the channels at different times. Once an anonymous group unsubscribes from an e-group, the sizes of the resulting channels could be different. Admittedly, rearranging the remaining subscribers within the e-group can make these channels equivalent again, but it may also cause NFS-related disunity within an anonymous group. Clearly, the cooperations of anonymous groups are necessary for keeping the equivalent channels at the same physical size (i.e. realizable NRS) and changing pattern (i.e. indistinguishable NFS).

In order to maintain the equivalency within every e-group, any anonymous group, who subscribes to equivalent channels, should be further modified such that the resulting G-NRS satisfy one of the following two theorems. The system will the final anonymized graph according to the G-NRS with the least modification cost.

Theorem 5.7. *If each channel in an e-group is subscribed by the same number of nodes from an anonymous group, the indistinguishability of the remaining subscribers is maintained when the anonymous group joins/leaves the channels.*

Proof. Assume that only anonymous group g changes subscriptions from t to $t+1$. Let $C = \{c_1, \ldots, c_x\}$ denote an e-group with x equivalent channels, and their edge ranks satisfy $\gamma(e_i^t) = \gamma(e_j^t)$ for $\forall c_i, c_j \in C$. When $\forall c_i \in C$ is subscribed by same number of g's members, we have $|\{v|c_i \in S(v,t), v \in g\}| = |\{v|c_j \in S(v,t), v \in g\}| = |g|/x$ for $\forall c_i, c_j \in C$. After g joins C, the ranks still satisfy $\gamma(e_i^{t+1}) = \gamma(e_j^{t+1}) = \gamma(e_i^t) + |g|/x$ (i.e. NRS indistinguishable). Since node can only subscribe to one channel in each e-group, after joining, $|S(v_i, t+1) \backslash S(v_i, t)| = |S(v_i, t+1) \backslash S(v_i, t)| = 1$ and $|S(v_i, t) \backslash S(v_i, t+1)| = |S(v_i, t) \backslash S(v_i, t+1)| = 0$ for $\forall v_i, v_j \in g$ (i.e. NFS indistinguishable). The leaving case can be proved in a same way; we skip that part. □

Theorem 5.8. *Given a set of anonymous groups that simultaneously join/leave an e-group, if each channel of the e-group always contains the same number of subscribers from this set, then the joining/leaving of these anonymous groups will not affect the* *existing privacy protection.*

Proof. Let $A = \{g_1, \dots, g_y\}$ denote a set of anonymous groups. Assume that only g_1 to g_y change their subscriptions from t to $t+1$. Like the proof in above, for the e-group, $\gamma(e_i^t) = \gamma(e_j^t)$. When $\forall c_i \in G$ has the same number of subscribers from A, we have $|\bigcup_{g \in A}\{v|c_i \in S(v,t), v \in g\}| = |\bigcup_{g \in A}\{v|c_j \in S(v,t), v \in g\}|$. When g_1 to g_y add subscriptions to C, the ranks still satisfy $\gamma(e_i^{t+1}) = \gamma(e_i^t) + |\bigcup_{g \in A}\{v|c_i \in S(v,t), v \in g\}| = \gamma(e_j^{t+1})$ for $\forall c_i, c_j \in C$ (i.e. NRS indistinguishable). Since g_1 to g_y always simultaneously join/leave C and each node can only subscribe to one channel from an e-group, the joining/leaving frequencies of the nodes in A are the same (i.e. NFS indistinguishable). $\square$

5.8 Performance Analysis and Evaluation

In this section, we conduct extensive experiments to evaluate the performances of our proposed schemes. To mimic the channel subscriptions of human beings, we first build a feature space with 5 dimensions. Each node and channel are randomly associated with some feature values. Based on the similarity of the features, we generate a threshold for each user channel pair. Consider that not all channels are popular. Even if the feature distances are the same, the thresholds may be different. During the simulation, at each time, we generate some random numbers; if the corresponding number is smaller than the threshold, we let the user subscribe to the channel; otherwise, the user has to leave it. The general setting of the synthetic data is as follows: there are $4,000$ nodes, 200 dummy node, 25 media channels, and the total number of time slices is 10. The experiments are conducted with MATLAB 2013b, running on a local machine with an Intel Core i7 Q720 and 12 GB RAM.

First, we check the importance of generating the K-anonymous for the whole temporal sequence. We compare our approach with the method, which only

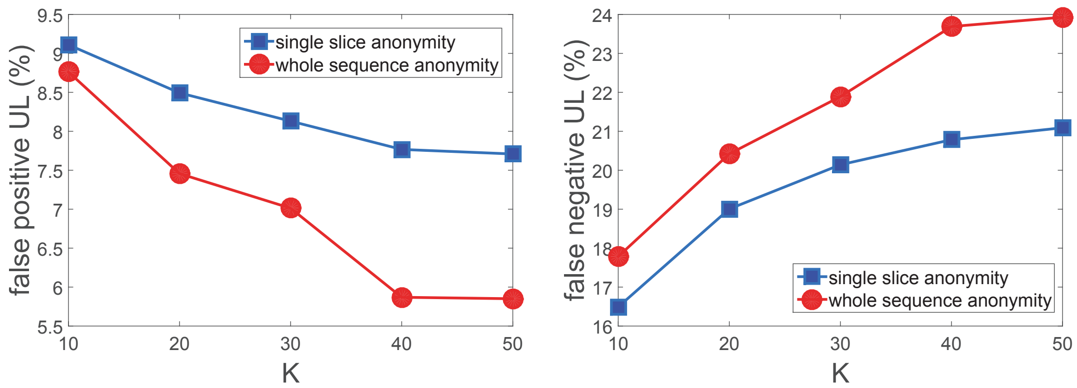
Table 5.1: The privacy risks analysis

Anonymous level	K=10	K=20	K=30	K=40	K=50
Whole sequence (%)	0.425	0.108	0.093	0.023	0.023
Slice-by-slice (%)	96.758	96.124	95.692	95.477	95.461

anonymizes a graph in each single time-slice. For quantifying the privacy, we follow paper [91]'s approach. The privacy risk of the whole graph is represented by $R(G)$, which is defined as: $R(G) = \frac{1}{|V|} \times \sum_{v \in V} \frac{1}{|A(v)|}$ where $A(v)$ is the anonymous group of node v, V represents all nodes on a given graph G, and $|\cdot|$ is the cardinality of a set. Table 5.1 shows the simulation result on an evolving graph with 5 slices. Clearly, the single slice-based anonymization loses most users' privacy.

Compared to the single slice-based anonymization, we further measure the extra utility loss of our approach in Fig. 5.3. The utility loss is calculated as the number of values that have been changed (i.e. $1 \to 0$ or $0 \to 1$) on the subscription tables for all time slices out of the total size of the tables. At a given time, if a user originally belongs to a channel, but the subscription is removed from the published anonymous graph, we call this situation false negative. Similarly, if there is no subscription on the original graph, but we add a new one due to anonymization, we call it a false positive. In Fig. 5.3, we compute the average false positive and false negative rate. From Fig. 5.3(a), we can see that the false positive rate of our approach is even lower than that of the single slice-based anonymization in 1% to 3%. As for false negative (i.e. Fig. 5.3(b)), our approach has about 1.5% to 3% more utility loss. Based on the results of Table 5.1 and Fig. 5.3, our approach sacrifices a small amount of utility but significantly improves the data privacy. Moreover, even for the slice-based approach, the overall subscription modification rate is around 25% to 30%. The ratio between the number of users and that of channels affects this basic modification rate.

Next, we consider the impacts of the subscription modification on node community features. When a pair of users simultaneously subscribes to the same channel, then



(a) False positive (b) False negative

Figure 5.3: Utility loss: whole sequence-based approach vs. single slice-based.

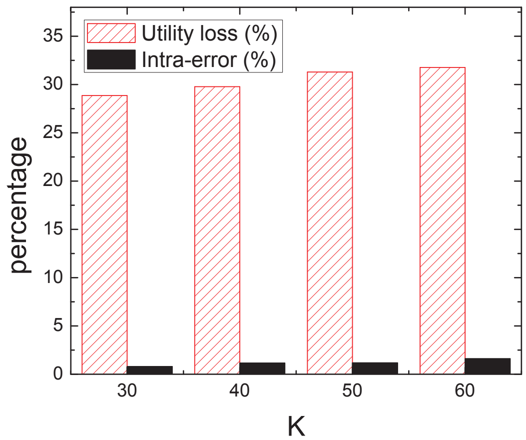
Table 5.2: The impacts of similarity metrics (Chapter 5.6)

Anonymous level	K=30	K=40	K=50	K=60	K=70
Specified compare (%)	37.89	37.90	37.92	37.93	37.93
Semi-specified (%)	29.11	31.46	31.66	33.19	33.42

we logically add a virtual edge between them since they share the same type of social interests. All the virtual edges form an interests-sharing graph, and in this part of the simulation, we test the edge modification rate of the interests-sharing graph before and after anonymization. On Fig. 5.4, although the overall modification rate of the original ESSN is around 30%, the impacts of anonymization on the interests-sharing graph is small.

In Chapter 5.6, we propose two similarity metrics for grouping similar nodes: a channel-semi-specific and a channel-specific. We test the impacts of using these metrics on the utility loss of the final published graph, which is shown in Table. 5.2. Since the data space is very sparse in our problem and nodes may have different dimensions, inappropriate grouping is easy to occur when using the channel-specific similarity metric. Therefore, it has more utility loss.

Since our model allows the usage of dummy nodes, we test their impacts in Fig. 5.5. During the simulation, we let the size of anonymous groups be at least 60, and take the subscription modification rate as utility loss. Since the tested number of dummy



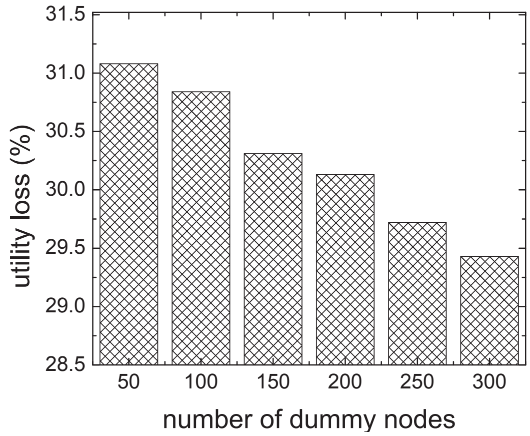


Figure 5.4: UL vs. intragroup error.

Figure 5.5: Impacts of dummy nodes.

nodes does not overwhelm the system, the utility loss rate is slightly reduced.

Fig. 5.6 tests the impacts of date space compression intensity on the execution time. When compressing and partitioning the original data space, the compression intensity partially influences the size of the partitioned regions since it changes the data points' clustering degree in the compressed space. With the growth of the compression intensity, the execution time first drops down and then slightly goes up. The reason for this phenomenon is that, at the low-intensity phase, the data space is still large and a lot of time is used for finding the data clusters, while at the high-intensity phase, more nodes may be put into the same region, which causes more execution time to be used for constructing the anonymous groups.

On an ESSN graph, the number of involved channels directly affects the hardness for constructing anonymous groups. The more channels that are considered, the fewer chances it has of having nodes with similar temporal subscriptions, which also means that more edge modifications will be used during anonymization. Moreover, for the same set of nodes, constructing a larger anonymous group will cause more utility loss. Fig. 5.7 is consistent with our expectation.

The basic idea of equivalent channels is to group several similar and unpopular

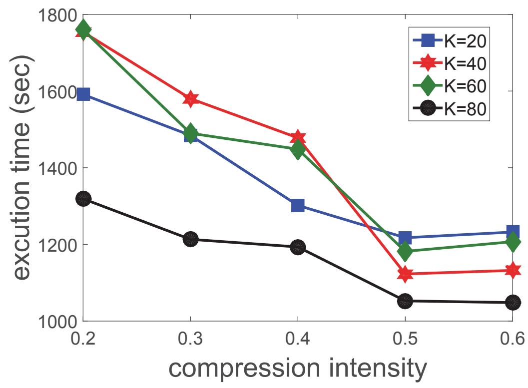


Figure 5.6: Exe. time vs. compression.

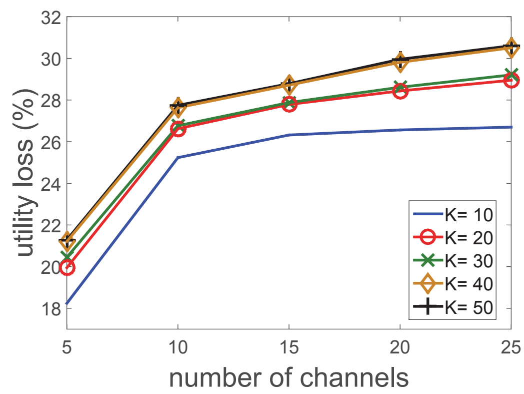
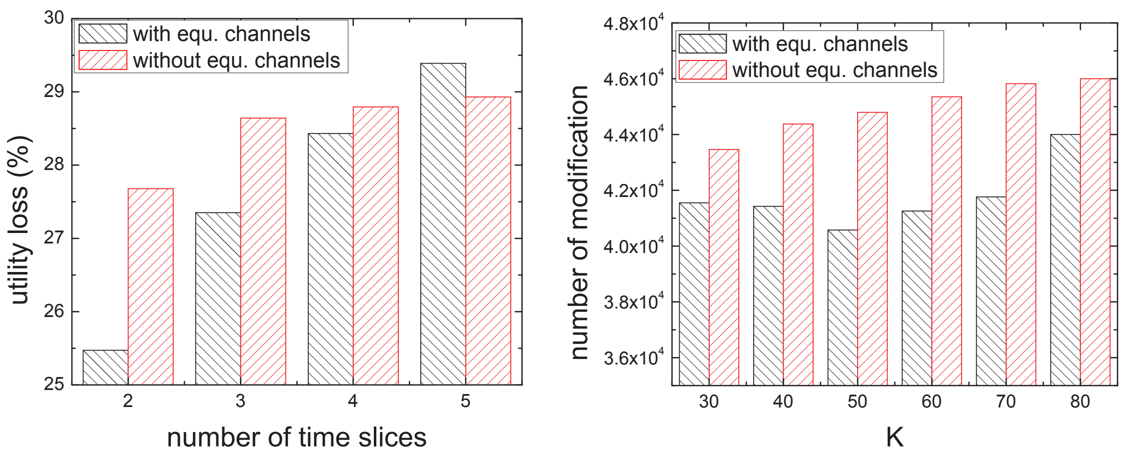


Figure 5.7: Impacts of channel number.

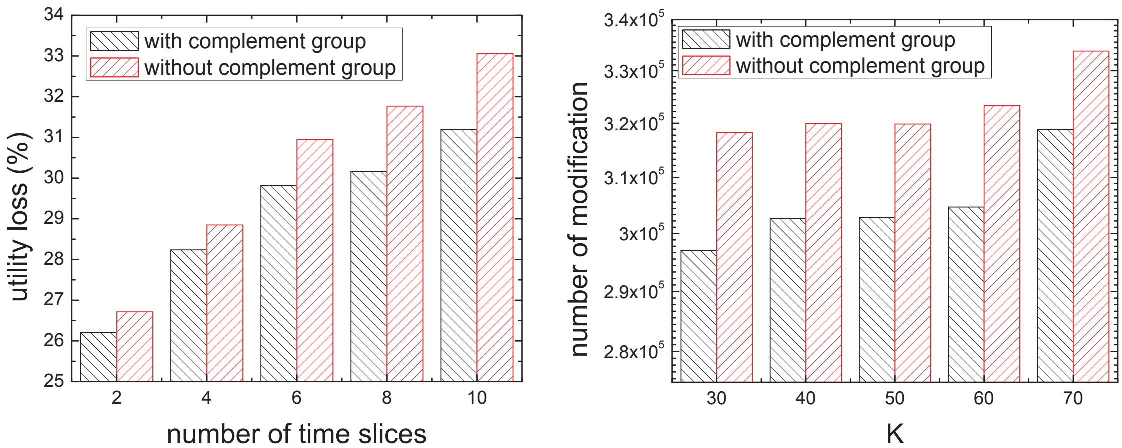


(a) time slice total number

(b) anonymous group size

Figure 5.8: The effects of equivalent channels.

channels together such that their subscription information can be partially preserved in an opportunistic manner. As shown in Fig. 5.8 (a), when an evolving graph contains a few temporal slices, creating the equivalent channels can save data utility. However, once the total number of the time slices is beyond a value, the uniqueness of these channels overwhelms their similarity, and establishing anonymous groups without equivalent channels saves more utilities. Fig. 5.8 (b) shows the absolution number of edge modification saved by using equivalent channels. It seems there is a trade-off between the total number of channels, the size of anonymous groups, and whether the equivalent channel is being used or not.



(a) time slice total number (b) anonymous group size

Figure 5.9: The effects of complementary anonymous groups.

Last, we check the impacts of using complementary groups (i.e. the anonymous groups satisfy Theorem 5.8) in Fig. 5.9. In order to guarantee that the members of an anonymous group have the same NRS and NFS, we must make sure that the node's joining/leaving frequency of each equivalent channel are the same. However, since most anonymous groups' members do not uniformly subscribe to the equivalent channels, having the concept of complementary groups gives more flexibility to the final subscriptions. From Fig. 5.9 (a) we can see that, once a system decides to use equivalent channels, establishing the complementary groups can always save utilities, and Fig. 5.9 (b) shows that using the complementary groups can save tens of thousands of subscriptions.

5.9 Chapter Summary

Over the past decade, we have witnessed an explosive growth of Online Social Network (OSN), and many applications have been designed by using certain social information. More and more institutions are willing to publish and analyze the OSN data.

Recently, people begin to consider the data publishing of time-varying social networks, also called evolving networks. Due to the increment of a time-dimension, the published data of an evolving social network is more vulnerable than that of the static one. In this chapter, we study the privacy-preserved data publishing of ESSN, and we propose a new type of attack, called the spatial-temporal attack. This attack can efficiently identify a target even if the evolving graph only contains a few time slices, and its background knowledge is easy to obtain.

Unlike conventional models, our problem requires the spatial-temporal anonymization on two types of information: subscribed channels' sizes, and frequencies of joining/leaving channels. The construction of such an anonymous group is challenging in three aspects. First, ESSN's data space exponentially grows with the number of channels, which makes the data points sparse. Second, changing a node's subscription will further cause the modification of other anonymous groups' channel size and frequency; there are consistency and realizability issues. Third, unpopular channel-related data are likely to be discarded during anonymization. In this chapter, we propose a three-step framework to solve the above issues.

The contributions of this chapter are as follows:

1. To our best knowledge, we are the first to study the anonymization of ESSN, under the spatial-temporal attacks. The proposed attack model is efficient, and the accessibility of the attack used background knowledge is the same as that of the traditional topology information.
2. We design several approaches to deal with the sparse data issue, high data discarding rate of unpopular channels, and the realizability of the final graph.
3. We conduct extensive experiments to show that our methods can efficiently provide spatial-temporal privacy protection with a reasonable amount of utility loss.

CHAPTER 6

PRIVACY-PRESERVED SOCIAL INFORMATION PUBLISHING WITHIN A DISTRIBUTED SYSTEM

In this chapter, we study the privacy-preserving data publishing problem in a distributed scenario. Many smartphone-based applications need microdata, which provides information at the level of individual respondents. Publishing a microdata table may leak respondents' privacy. Conventional research on privacy-preserving data publishing focuses on providing identical privacy protection to all data requesters. But, in social networks, information usually propagates from friend to friend, instead of being trapped in a small coterie. As a result, multi-level privacy protection should be provided: along a data propagation path, a series of tables will be locally created at each participant, and the tables' privacy-levels should be gradually enhanced.

However, the tradeoff between the created tables' overall utility and their individual privacy requirements are not trivial, especially for distributed schemes: any improper sanitization operation under a lower privacy requirement may cause dramatic utility loss on the subsequent tables. In order to solve the problem, an approximation algorithm is designed by previewing the future privacy requirements during data propagation. Social-crowdsourcing system, which is introduced in Chapter 3, can directly adopt this chapter's data anonymization algorithm if the privacy issue is considered and the social information used in social-crowdsourcing is propagated via multi-hop relays.

6.1 Introduction

Learning others' social features can significantly improve the performance of many mobile social network-related tasks, such as data routing [92], personalized recommendation [93] and social relationship prediction [94]. In these scenarios, a participant needs access to a large volume of personal information in order to spot the pattern [95]. A dataset, which consists of the information at the level of individual respondents, is known as microdata dataset. In order to protect the privacy of each individual respondent, data holders must carefully sanitize (also known as anonymize) the dataset before publishing. In the past decade, many privacy standards have been proposed, such as k-anonymity [96], l-diversity [97], and t-closeness [98].

Unlike the conventional centralized database system, where data requesters *directly* interact with data owners, information on a mobile social network is disseminated from user to user via *multi-hop relays*. Considering the well-known limitations with centralized systems, such as system bottlenecks or a single point of attacks problem, in this chapter, we study the problem of multi-hop relay-based privacy-preserving data publishing, where a microdata table is gradually propagated from its original owner to distant people. However, under this scheme, the recipients will present different trust levels regarding the original data owner. Intuitively, after each time of relay, one should further provide more privacy protections on the data. For example, in Fig. 6.1(a), along a social path with length K, each user eventually will get one copy of v_0's table, and we need the tables' privacy to be gradually reinforced, as shown in Fig. 6.1(b). Data privacy and data utility are naturally at odds with each other [95]: The more privacy a dataset preserves, the less utility the dataset has. This propagation scheme creates a unique problem: 'for a group of friends, how can they

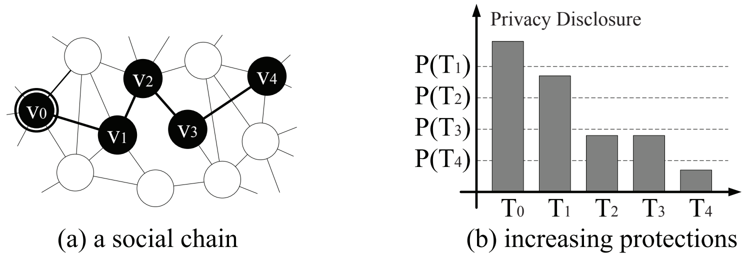


Figure 6.1: An example of the target problem. Black nodes create a social chain with length 4, dash lines represent the enhanced privacy requirements, and bars stand for the real privacy value of the table.

create a series of tables with maximal overall data utility, and assure that the tables' privacy are increasingly protected at the same time?' To our best knowledge, this unique problem has never been proposed or solved.

Take Fig. 6.2 as an example. Suppose that l-diversity is the privacy requirement, and the total target propagation distance l is equal to 2. Assume that the corresponding participants are v_0, v_1, and v_2. With the growing of the propagation distance, the parameter l becomes larger and larger (i.e. after the first hop, the table should satisfy 2-diversity, and after the second hop, it should satisfy 3-diversity). The original dataset is given by Fig. 6.2(a). Figs. 6.2(b) and (c) give the results by directly using anonymizing operations on the original table T_0. We can see that the sanitized values are different in these two tables. However, during multi-hop relays, a participant can only observe the table passed from the previous one, and therefore, if v_0 gives T_1 to v_1, v_2 can only obtain Fig. 6.2(c), instead of T_2. Consider that the tables, which satisfy (l+1)-diversity, must satisfy l-diversity. For v_0, he has two options for sending the dataset to v_1: he either sends T_3 or T_2. For the first case, the user v_1 only needs to forward T_3 to v_2 without any changes, while for the other case, v_1 should further sanitize T_2 and send the result T_3 to v_2. Clearly, we cannot simply claim which approach is better; this is because it depends on the utility value of each attribute. For instance, if we define that any suppression operation costs 1 unit of utility, then the first option loses a total of 12 units of utility (as 6 units of utility are lost for T_1), while the second one costs 16 units (as 4 utility units are lost for T_2 and 12 for T_3). However, if the age attribute is more important than the zip code, assuming that suppression on age costs 2.5 units of utility and suppression on zip code costs 1, the utility loss of the second option becomes 28, while that of the first option is 30.

In order to solve the problem, we first propose two greedy schemes: a bottom-up algorithm and a top-down algorithm. The bottom-up algorithm works at the tuple-level, and it always merges similar tuples into one equivalent class until all classes meet the corresponding privacy requirement. The top-down approach is based on the observation that the larger an equivalent class is, the safer its members are. This approach considers how to split a table into the maximum number of sub-tables, such that each small table has the required privacy protection. By comparing the results of both algorithms, we propose another approximation algorithm for optimally creating a series of tables. Consider that if a table satisfies a higher privacy requirement, then it also meets the lower one. At each participant, the algorithm does not simply maximize the data utility at its required privacy-level; instead, it takes future, more stringent privacy requirements into consideration. Extensive simulation results show that more data utilities are preserved by adopting our scheme.

6.2 Literature Review

Privacy-preserving data publishing has received considerable attention in the past decade. Based on the different goals of privacy-preservation and data features, a variety of privacy protection techniques have been designed. Statistical databases are used for the purpose of statistical analysis, and it aims to provide aggregated knowledge. Since intelligent users may use a combination of aggregate queries to derive information about an individual record, statistical databases need privacy

Name	Age	Zipcode	Disease
Ashley	23	19024	Hepatitis
Brooke	23	19024	Brochitis
Charish	28	19024	Flu
Dave	28	19122	Cancer
Ellen	29	19122	Hepatitis
Frank	24	19122	Brochitis

(a) T0: Patient Data (original)

	Age	Zipcode	Disease
1	23	19024	Hepatitis
2	23	19024	Brochitis
3	28	19***	Flu
4	28	19***	Cancer
5	2*	19122	Hepatitis
6	2*	19122	Brochitis

(b) T1: 2-diversity table

	Age	Zipcode	Disease
1	2*	19024	Hepatitis
2	2*	19024	Brochitis
3	2*	19024	Flu
4	2*	19122	Cancer
5	2*	19122	Hepatitis
6	2*	19122	Brochitis

(c) T2: 3-diversity table

	Age	Zipcode	Disease
1	2*	19***	Hepatitis
2	2*	19***	Brochitis
3	2*	19***	Flu
4	2*	19***	Cancer
5	2*	19***	Hepatitis
6	2*	19***	Brochitis

(d) T3: 3-diversity table (based on T1)

Figure 6.2: Generated tables based on requirements with different privacy levels.

protection. Commonly used privacy-preservation techniques in this field include adding random noise while maintaining certain statistical features [99, 100], and making perturbation on the query-results [101, 102]. Differential privacy [103, 104] aims to provide an indistinguishability regarding the existence of an individual respondent, during interactive queries. Unlike traditional approaches, differential privacy puts restrictions on the release mechanism instead of the original dataset.

Unlike a statistical database, a microdata database publishes information at the level of individual respondents [105], and its records are unperturbed. In order to preserve individual respondents' privacy, many privacy metrics have been developed, such as k-anonymity [96], l-diversity [97], and t-closeness [98]. In order to achieve these metrics, certain anonymizing operations are adopted; generalization and suppression [106, 107] to quasi-identifiers are the most two commonly used operations. Essentially, the anonymizing operations divide tuples of a table into multiple disjoint groups, and the tuples from the same group form an equivalence class. k-anonymity requires that the size of each group must be at least k; l-diversity needs each group to contain at least l unique sensitive values, and t-closeness requests the distribution of the sensitive attributes within each group to be similar to the distribution within the original dataset. In order to maximize the anonymized table's utility, many top-down or bottom-up approaches [108, 109, 110, 111] have been proposed based on different anonymization requirements. However, almost all of the existing studies focus on providing a single privacy-preserving level, while, in practice, the recipients are not equally trustable [112], especially on a social network. To our best knowledge, only papers [112, 113] consider multi-level privacy preservation on statistical databases. Due to the differences about the goal of privacy-preservation and anonymizing operations, their approaches cannot be applied to microdata tables. Moreover, they use a centralized scheme, but our problem considers a distributed scenario.

The cost of privacy is the loss of data utility. For a centralized database system, before publishing a table, the data holder first defines a privacy requirement, and then, anonymizes the table with the minimal utility damage. Certain works [95, 114, 115] have been done on this kind of tradeoff between privacy and utility. However, few people have worked on the privacy-utility tradeoff about *multiple related tables*. Unlike the traditional database, propagating a dataset along a social path with enhancing privacy-preserving requirements essentially creates multiple mutually-related tables. In practice, it is often necessary to maximize the overall utilities of the whole tables, instead of simply optimizing one table's utility. Also, the optimization of overall utility is usually not equivalent to the simple combination of individual tables' optimal results, as we have shown in Chapter 6.1.

6.3 Problem Model and Formulation

Conventional privacy-preserving data publishing problems focus on providing uniform privacy protection to all requesters, and therefore, in their models, the database is the only source of information. However, in real life, information usually

Table 6.1: Notations used in this chapter

Notation	Description
δ	a pre-defined privacy threshold, $\delta \in (0, 1)$.
l	the length of propagation.
m	the total number of tuples.
n	the number of attributes.
v_i	the ith node on a social chain, $i \in [0, l]$.
T_i	the database table possessed by v_i.
t_i	the jth tuple (row) in a table, $i \in [0, m]$.
$\langle t \rangle$	an equivalent class of tuple t.
A_i	the quasi-identifiers, $i \in [1, n-1]$.
A_n	the sensitive attribute.
$\sigma_i(T_j)$	select tuple t_i from table T_j.
$\pi_i(T_j)$	project the quasi-identifier value A_i in T_j.
w_i	utility weights for quasi-identifiers.
$p(\cdot)$	the distribution of a given set of values.
$P(T)$	table T's privacy disclosure function.
$U(T)$	table T's data utility function.
$JS(\cdot, \cdot)$	JensenShannon divergence function.
$d(\cdot, \cdot)$	sematic distance between two values.
$\|v_i - v_0\|$	social distance from v_i to v_0.
$H(A_n\|T)$	information gain about sensitive value A_n given T.

propagates from person to person in a distributed way, instead of being trapped in a small coterie. In this chapter, we propose a distributed privacy-preserving scheme, which provides a multi-level privacy for different social users. The most significant feature of the proposing scheme is that the data's privacy is further strengthened after each time of dissemination. In this section, we present the problem formulation, system model, and some commonly used symbols.

6.3.1 Basic Model

On a social network, there is a social chain consisting of $l + 1$ participants (also called nodes), assuming $v_0, v_1, \ldots, v_l$. Let $T = \{t_1, t_2, \ldots, t_m\}$ be a microdata table of m tuples and n attributes $A = \{A_i\}$. Usually, we have $m >> n$. In practice, a microdata table may contain several sensitive attributes, such as salary and disease.

Since we can regard each possible combination of the sensitive attributes as a data point (vector) in a hyper-dimension, here, we assume that there is only one sensitive attribute, which is A_n. In other words, attributes $\{A_1, A_2, \ldots, A_{n-1}\}$ are the quasi-identifiers (non-sensitive attributes), and A_n is the sensitive attribute.

Definition 6.1. In a table T, tuples with the same quasi-identifiers' values form an equivalent class $\langle t \rangle$.

For example, in Fig. 6.2(b), $\langle t_3 \rangle$ means the equivalent class of the third tuple, and therefore, $\langle t_3 \rangle$=$\langle t_4 \rangle$= {(28, 19***, Flu); (28, 19***, Cancer)}. Also, in Fig. 6.2(c),$\langle t_4 \rangle$=$\langle t_5 \rangle$=$\langle t_6 \rangle$= {(2*, 19122, Cancer); (2*, 19122, Hepatitis); (2*, 19122, Brochitis)}.

Let σ be the selection operation from a table, and $\sigma_i(T_j)$ stands for the ith tuple of table T_j. For instance, in Fig. 6.2(b), $\sigma_3(T_1)$ means selecting the third tuple from T_1, and therefore, $\sigma_3(T_1)$={(28, 19***, Flu)}. In Fig. 6.2(c),$\sigma_3(T_2)$= {(2*, 19122, Cancer)}. We use π as the projection operation on a table. $\pi_i(T)$ gives the ith column of table T. For instance, in Fig. 6.2(a), $\pi_3(T_0)$={Hepatitis; Brochitis; Flu; Cancer; Hepatitis; Brochitis }. Since we assume that the nth attribute is the sensitive one, $\pi_n(T)$ means the sensitive values in T. For the ease of description, for the rest of the chapter, let $\pi(T)$ be the short of $\pi_n(T)$. So, $\pi(T_0) = \pi_3(T_0)$. In addition, any value in table T_k, attribute (column) A_j, tuple (row) t_i is represented as $\sigma_i(\pi_j(T_k))$. In Fig. 6.2(b), $\sigma_3(\pi(T_1))$ ={Flu}, and $\sigma_{\langle 3 \rangle}(\pi(T_1))$ ={Flu, Cancer}.

Along the social chain, v_0 is the original data owner, and based on trust relations, user v_i ($i \in [1, l]$) receives an anonymized table T_i from his previous user and sends a modified version, T_{i+1}, to the next one. For instance, v_0 sends table T_1 to v_1; based on T_1, the receiver, v_1, creates table T_2 and sends it to v_2, and so on. Essentially, each user locally builds a table based on previous user's data, and therefore, T_i is a sub-table of T_{i-1}. Considering the fact that trust fades along a social path, instead of having l tables with the same privacy-preserving strength, our system builds a series of tables with *increasing* privacy protections. Therefore, table T_{i+1} is the sanitization result of table T_i. For benefiting the whole society on a social chain, the data owner wants to provide the maximal *overall* utility of these tables.

Definition 6.2. Given a pair of tables T and T', if the value of any tuple in T is covered by the value of T''s corresponding tuple, $\forall i \in [1, l], \forall j \in [1, m], \sigma_i(\pi_j(T)) \subseteq \sigma_i(\pi_j(T'))$, then we call T' a *sanitization result* of T.

For example, value "19**2" covers "191*2", since the value's range of "191*2" is a subset of the former one. Note that any value covers itself, e.g. "19**2" covers "19**2."

In practice, different attributes may have different levels of importance. For example, for the data published by Screen Actors Guild, the public cares more about the actor/actress's age or weight than their academic performance. In our model, we classify users into several types, and for the tables owned by each type of user, there is a set of utility weights $W = \{w_1, w_2, \ldots, w_{n-1}\}$ associated with quasi-identifiers ($w_i \in [0, 1], \sum_i w_i = 1$). The larger the weight is, the more important the attribute is. Utility weight is a *source-based* concept: along a table's propagation path, all participants use the same utility weights since the tables have the same source.

6.3.2 Problem Formulation

6.3.2.1 Privacy Disclosure Measurement

Let $p(\cdot)$ denote the distribution of a given set of values, then $p\left(\pi(T)\right)$ represents the sensitive values' distribution over the whole table T, and $p\left(\sigma_{\langle t\rangle}(\pi(T))\right)$ indicates that distribution within an equivalent class $\langle t\rangle$. For a given table T, its privacy disclosure $P(T)$ is measured as the maximum amount of privacy disclosure from its equivalent classes:

$$P(T) = \max_{\forall \langle t \rangle \subseteq T} JS\left(p(\pi(T)), p(\sigma_{\langle t \rangle}(\pi(T)))\right)$$

where $JS(\cdot,\cdot)$ is Jensen-Shannon divergence function [116], which measures the similarity between two distributions. $JS \in [0,1]$. The smaller the JS is, the closer the distributions are.

6.3.2.2 Data Utility Measurement

Assume that $\sigma_i(\pi_j(T))$ and $\sigma_i(\pi_j(T'))$ are the same data point appearing in two tables. Due to the sanitization operations, the values of them may not be same. Let $d\left(\sigma_i(\pi_j(T)), \sigma_i(\pi_j(T'))\right)$ be the semantic distance between them. For instance, by using suppression operations, the distance between "19*22" and "19***" is 2 units. If we use generalization, the distance between age "29" and range "[20-30)" is defined as the distance from 29 to the mean age within that age group. For a given attribute, if its values are extremely similar to each other, basically, the whole attribute becomes useless for most of the data mining task. As a result, the utility of an attribute mainly comes from its diversity. If we use sanitization operations to merge all tuples of table T into one equivalent class, which has the same effects as only publishing sensitive attributes, then the resulting table $\widetilde{T}$ reaches maximum privacy but minimum utility. For the rest of the chapter, we use $\widetilde{T}$ to represent the resulting table by sanitizing a given table T into one equivalent class. Let $U(T)$ stand for the data utility of any table T, then we have:

$$U(T) = \sum_{i=1}^{n-1} \left[w_i \sum_{j=1}^{m} d\left(\sigma_j\left(\pi_i(T)\right), \sigma_j(\pi_i(\widetilde{T}))\right) \right]$$

6.3.2.3 Formulation and Features

By hiding the exact value of attributes, a table can gain more privacy but might lose unity. Along a table's propagation path, the privacy protection on the table should be gradually enhanced, since the trust relations between the original data owner and each receiver is fading. As a result, before disclosing a table to the next recipient, each relay node must further anonymize the table until it reaches a certain privacy level. Here is the problem formulation: for a given table T_0, along a social path with length l, we want to create l tables; each table T_i is the sanitization result of T_{i-1} ($i \in [1, l]$), and these tables satisfy the following requirements:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{l} U(T_i) \\ \text{Subject to} \quad & P(T_i) < (l - i + 1)\delta, \ \text{for } \forall i \in [1, l] \end{aligned}$$

Theorem 6.3. *For a given microdata table T_0 and a pair of source-selected parameters δ, l, there is at least one series of tables $T_1, T_2, \ldots, T_l$ satisfied the increasing privacy requirements $P(T_i) < (l - i + 1)\delta$ for $\forall i \in [1, l]$.*

Proof. One can delete all attributes of T_0 except the sensitive ones, and let the result $\widetilde{T}$ be T_i. On table T_i, all tuples form only one equivalent class, and therefore $P(T_i) = \max_{\forall \langle t \rangle} JS\left(p(\pi(\widetilde{T})), p(\sigma_{\langle t \rangle}(\pi(\widetilde{T})))\right) = 0$. Since $(l-i+1)\delta > 0$ for $\forall i \in [1, l]$, the created tables, $T_1, T_2, \ldots, T_l$, always satisfy the privacy requirements. $\square$

Theorem 6.4. *Along the tables' propagation path, the table utility U is non-increasing: $U(T_i) \geq U(T_j)$, for $\forall i, j, 1 \leq i < j \leq l$.*

Proof. Because sanitization operations (i.e. generalization and suppression) are irreversible, for $\forall i, j, 1 \leq i < j \leq l$, table T_j must be a child-table of table T_i, and therefore, its utility should be less than T_i. Even if there exists a pair of tables

T_i' and T_j', whose table utility satisfies $U(T_i') < U(T_j')$, one can simply replace T_i' with T_j' because $P(T_j') < (l-j+1)\delta < (l-i+1)\delta$. $\square$

Theorem 6.5. *The target problem: "given table T_0, propagation length l, and a threshold δ, maximize $\sum_{i=1}^{l} U(T_i)$ subject to $P(T_i) < (l-i+1)\delta$ for $\forall i \in [1,l]$" is NP-hard.*

Proof. We want to reduce our original problem to the classic optimal L-diversity problem, which is NP-hard [117]. Consider the simplest case of our problem, where $l = 1$. Under this special case, our problem is reduced to: given table T_0 and a threshold $\delta > 0$, Maximize $U(T_1)$ subject to $P(T_1) < \delta$, where T_1 is a sub-table of T_0. Since the amount of privacy disclosure is measured by Jensen-Shannon divergence, each type of sensitive value must appear at least once within each equivalent class. Assume that there is a total of L types of sensitive values. After we further simplify the problem into the situation that all sensitive values follow a uniform distribution, the problem becomes a typical L-diversity problem. As a result, our target problem is NP-hard. $\square$

As Theorem. 6.5 shows, the target problem is NP-hard even if a table is propagated only one hop. Actually, maximizing the overall utility of l tables is more challenging than the one-hop case. Since the anonymizing operations are irreversible, once we put a tuple's value into a broader range via sanitization operations, for the remaining participants in downstream, no one can change it back, or even put it in a narrower range. Therefore, the processing effects of previous participants are accumulated.

Just maximizing a table's utility subject to a current privacy requirement is not a good option: under a lower privacy requirement, merging a pair of tuples may achieve a higher utility, but later, in a higher privacy condition, it turns out that they should be placed in different equivalent classes. This problem not only happens when using our privacy metric but also exists when using the classic l-diversity or

t-closeness standards. For example, in Fig. 6.2, tuples 3 and 4 are closer to each other, and it is optimal to put them together for 2-diversity conditions. Once it is done, their attributes become the same except for the sensitive attribute column. However, distances from the modified tuples to their second closest tuples are increased (e.g. the distance between tuples 2 and 3 in T_1), which may cause extra utility losses for creating tables with a higher privacy requirement. Clearly, when sequentially creating a series of tables with increasing privacy protection, tuples grouping orders affect the overall utilities.

6.4 Key Observations

Observation 6.6. After sanitization, each type of sensitive values must appear at least once in each equivalent class.

According to the definition of our privacy disclosure function $P(\cdot)$, if an equivalent class does not contain a sensitive value, its Jensen-Shannon divergence distance becomes undefined, which breaks the privacy requirement of our problem: $P(T_i) < (l - i + 1)\delta$. Note that Jensen-Shannon divergence is based the Kullback-Leibler divergence. If one type of value appears zero times in one distribution, then we will face the "Division by zero" problem.

Observation 6.7. In general, the larger a subset of records is, the more likely that its sensitive values' distribution is close to the global one.

Based on Observation. 6.7, we infer that, if the privacy-preserving level of a set of records is far beyond the current requirement, one can partition the set into multiple subsets to reduce each subset's privacy-preserving level; in the meanwhile, the utility loss of using sanitization operations on the given records can also be reduced.

Observation 6.8. For a given set of records, whose sensitive values' occurrence times follow a certain distribution, if we partition the given set into several subsets such that the sensitive values' occurrence frequencies in each subset are unchanged, then the amount of privacy disclosure of each subset is the same as the original set's privacy disclosure.

When we use suppression/generalization on a set of records, the larger the set is the more data utility that is lost. After finding a set of records satisfying a given privacy requirement, one can further reduce the cost of sanitization operations by dividing the set into smaller ones while keeping the distribution of sensitive values unchanged.

Observation 6.9. For a given set of records, one can create several subsets by letting each subset uniquely dominate on one or more sensitive value types. The merging of these subsets may reduce the statistical distance between their sensitive values' distribution and the global one.

For instance, given a set of records, whose sensitive values' occurrence times are $3:5:7:9:11$, we can create two subsets with sensitive values' occurrence times $1:1:6:1:1$ and $1:1:1:8:1$, respectively. The JS distances of these subsets to the original set are 0.6137 and 0.5994. After merging sets into one equivalent class, the occurrence times of sensitive values become $2:2:7:9:2$, whose JS distance becomes 0.3075.

6.5 Solution Details

This section consists of three parts. In part one, we propose a bottom-up table sanitization approach. The main idea of the bottom-up approach is to merge similar tuples into an equivalent class until the table meets the privacy requirement. By tracking any tuple's locations in the table series, we can see that, with the growth of privacy-preserving level, the size of the tuple's resided equivalent class becomes larger and larger. In the second part, we propose a top-down table sanitization approach.

Unlike in the previous approach, the top-down approach gives more concern on the table creation at higher privacy levels. The intuition of the second approach is to gradually partition tables into multiple clusters until any further partitioning will break the privacy requirement. In order to save more data utility, after reaching the required privacy-level, both the bottom-up approach and top-down approach split the equivalent classes into smaller ones by using Observation 6.8's feature. In the last part, we first compare the above two schemes, and then propose another approach by considering the utility loss at future higher privacy requirements. Note that the first two algorithms only consider the instant privacy requirement (single level), while the last one takes both current and future privacy requirements (multiple levels) into consideration.

6.5.1 Bottom-Up Approach

The Bottom-Up Approach (BUA) treats each record individually, and the table creating process by BUA is similar to the changing pattern of tuples' grouping during multi-hop relays: initially, there are lots of small-sized equivalent classes, and gradually, more and more classes merge into others; with the growth of the sizes of equivalent classes, the distribution of the sensitive values within each equivalent class becomes closer to the distribution over the whole table.

There are two objections with our problem: on the one hand, we need to maximize a table's remaining utility after sanitization operations, and on the other hand, the amount of privacy disclosure of the resulting table should be bounded. Since privacy and utility are not comparable with each other, we cannot consider these two objections at the same time. In order to meet the utility objection, one can merge similar tuples into one equivalent class, as many as possible, until the class reaches the required privacy-level. For satisfying the privacy, one can group the tuples according

Algorithm 15 Bottom-Up table sanitization Approach (BUA)

Input: Privacy requirement threshold δ and original table T
/*Phase I: Create sensitive-value dominating groups*/
Create basic group set $\{g\}$ by including each type of sensitive value once
Compute semantic distance d from any un-grouped tuple to $\{g\}$
for Each basic group $g_i \in \{g\}$ **do**
  for Each type of sensitive value $a_i \in A_n$ **do**
    Compute the overall distance from g_i to the tuples with a_i
Create data fragments $\{f\}$ dominating one sensitive value type.
/*Phase II: Group combinations*/
while $\exists f_i \in \{f\}$, whose $P(f_i) > \delta$ **do**
  for Each fragment $f_j \in \{f\}$, and $f_j \neq f_i$ **do**
    Compute the distance $d(f_i, f_j)$ for merging f_i and f_j
  Sort $d(f_i, f_j)$ in ascending orders
  for Each value in $d(f_i, f_j)$ **do**
    if $P(\{f_i, f_j\}) < P(f_i)$ **then**
      merge f_i and f_j into one equivalent class
Create grouping index IDX based on $\{f\}$
/*Phase III: Further partitioning based on Observation 6.8*/
Call Algorithm 16, $IDX \leftarrow PLSA(\delta, T, IDX)$
Output: The grouping index set IDX

to the occurrence frequency of the sensitive values. For example, if the whole table's occurrence frequencies of the sensitive values are 20% : 40% : 40%, then when creating an equivalent class, the operator should merge at least 5 tuples in each time, including one tuple with sensitive value type I, two tuples with type II, and another two tuples with type III. However, neither of the approaches can create a table with maximum utility and satisfy a given privacy-preserving level.

The basic idea of BUA is as follows: at the lower privacy-preserving requirements, if each equivalent class dominates in one type of sensitive value, the merging of different classes may make their sensitive values' distribution closer to that of the whole table. BUA consists of two phases. Considering that each type of sensitive values must occur once in each equivalent class, in phase I, BUA first creates several basic groups, where each sensitive value appears once within each group. For the remaining records, we compute their quasi-identifiers' distance to each group. BUA proportionally assigns each basic group with a dominating sensitive value type, according to the occurrence

Algorithm 16 Privacy Level-preserved Splitting Algorithm (PLSA)

Input: Privacy requirement threshold δ, original table T, and grouping index IDX
Setup: operating table set $S \leftarrow IDX$ and $IDX \leftarrow \phi$
for $S \neq \phi$ **do**
  Fetch table T' from S, $S \leftarrow S \setminus \{T'\}$
  for Each type of sensitive value $a_i \in A_n$ in T' **do**
    Assign tuples with a_i into two groups S_{a_i} and S'_{a_i}
  Create subtables T_1, T_2 by optimally combining $\{S_{a_i}\}$, $\{S'_{a_i}\}$
  if $P(T_1) < \delta$ and $P(T_2) < \delta$ **then**
    Update the operation set $S \leftarrow S \bigcup \{T_1, T_2\}$
  else
    Update the index set $IDX \leftarrow IDX \bigcup \{T'\}$
Output: The grouping index set IDX

frequency of sensitive values. Based on tuple's sensitive value and its distance to groups, each tuple joins a group. In phase II, BUA gradually merges groups in order to satisfy a given privacy requirement, and the merging process will not terminate until all equivalent classes reach the required privacy-level. Algorithm 15 gives the procedure of BUA.

Essentially, by using BUA, along the propagation path, each node tries to locally maximize its table's utility subject to the privacy requirement. However, the process may cause inappropriate grouping, especially when creating the basic groups. Take Fig. 6.3 as an example. S_1, S_2 and S_3 are three large sets of data points, and V is a small set. Distance d represents difficulty for merging the corresponding data into an equivalent class, and $d_1 < d_2 < d_4 < d_1 + d_3$. In Fig. 6.3(a), initially data point V is slightly closer to equivalent class S_1 than class S_2. At a lower privacy requirement (Fig. 6.3(b)), BUA should merge V with S_1, and in a later time (Fig. 6.3(c)), the S_1 will merge with another equivalent class S_3. However, if we directly increase the privacy requirement from 3δ to δ, the data point V will join the S_2 set. In short, whether a node joins an equivalent class is determined not only by the attribute distance between the node and the class but also the distance between the node and other classes at which the class is going to merge at higher privacy requirements.

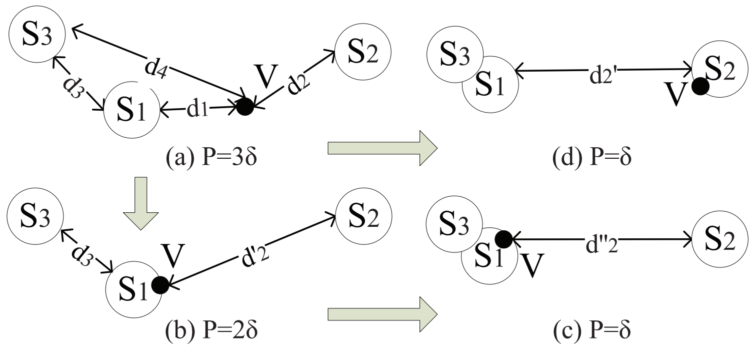


Figure 6.3: The grouping problem with BUA.

Algorithm 17 Top-down table sanitization Approach (TDA)

Input: Privacy requirement threshold δ and original table T
/*Phase I: Clustering until any further clustering violates δ*/
Set up the operation table set by $S \leftarrow \{T\}$
Set up output equivalent class index set by $IDX \leftarrow \phi$
while $S \neq \phi$ **do**
  Fetch a table T' from S, $S \leftarrow S \setminus \{T'\}$
  Create 2 clusters T_1 and T_2: $T_1 \bigcap T_2 = \phi$ and $T_1 \bigcup T_2 = T'$
  if $P(T_1) < \delta$ and $P(T_2) < \delta$ **then**
    Update the operation set $S \leftarrow S \bigcup \{T_1, T_2\}$
  else
    Update the index set $IDX \leftarrow IDX \bigcup \{T'\}$
/*Phase II: Further partitioning based on Observation 6.8*/
Call Algorithm 16, $IDX \leftarrow PLSA(\delta, T, IDX)$
Output: The grouping index set IDX

However, BUA does not consider the impacts of further merging operations.

6.5.2 Top-Down Approach

Top-down Approach (TDA) considers how to partition a table into multiple sub-tables such that each sub-table satisfies a single-level privacy requirement. The basic idea of TDA comes from paper [118], which provides k-anonymity to data, but our problem requires a distribution-related anonymity. The TDA is based on two insights. First, for a given set of data $\{S\}$, the distribution of which follows $D(\{S\})$, if we partition the set $\{S\}$ into two subsets ($\{S_1\}$, $\{S_2\}$) and keep each value's occurrence frequency unchanged in both of the subsets, then we have $D(\{S_1\}) \approxeq D(\{S\})$ and

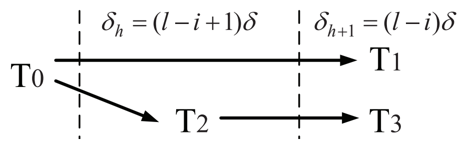


Figure 6.4: Forward looking the future privacy requirement.

$D(\{S_2\}) \approxeq D(\{S\})$. This feature maximizes the protection of privacy but hurts data utility. The second insight is that, if we randomly partition a table into several sub-tables, the larger a sub-table is, the more likely that the distribution of a sensitive attribute in the sub-table is close to the distribution of the attribute in the original table.

Based on these two insights, we propose our TDA, as shown in Algorithm 17. The algorithm consists of two phases. The first phase gradually partitions the original data table into several sub-tables. This part tries to maximally preserve data utility and satisfy the needs of privacy. Note that, during the process of the table's propagation, once a pair of records been merged into one equivalent class at a node, it is hard for the following nodes to take them apart since the following nodes cannot discriminate the tuples from the same equivalent class. Based on this consideration, instead of partitioning the original table at one time, we first divide the table into two parts, then we split each sub-table into another two parts, and so on. For any sub-table, the partitioning process stops when the further partitioning on it will break the privacy requirement. Due to the fact that the smaller an equivalent class is, the more data utility preserves, the second phase tries to further reduce the size of each sub-table T, and in the meanwhile, keeps their privacy-preserving degree unchanged by using Algorithm 16.

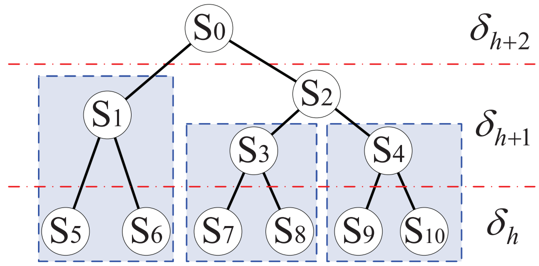


Figure 6.5: The idea of FLA. $\delta_{h+2} < \delta_{h+1} < \delta_h$.

6.5.3 Forward Looking Approach

In order to achieve the optimization of the overall utility of tables, we propose another algorithm called Forward Looking-based table sanitization Approach (FLA), which improves the solutions from two aspects. For reducing the impacts of locality, instead of only considering the current privacy requirement, FLA takes the next privacy-level into account: FLA creates two tables T_i and T_{i+1} such that $U(T_i) + U(T_{i+1})$ is maximized subject to $P(T_i) < (l-i+1)\delta$ and $P(T_{i+1}) < (l-i)\delta$.

FLA also improves the results from the methodology. The advantage of BUA is that, by grouping tuples into fragments dominating on certain sensitive values, one can easily create tables with different privacy levels. However, BUA's main problem is caused by the inappropriate merging at the lower privacy requirements, which results in lots of utility loss at the conditions with higher privacy requirements. On the other hand, TDA can easily group tuples at higher privacy requirements, but its phase II lacks flexibility: any partition on a sub-table will not change the sensitive values' distribution, which potentially causes utility loss.

FLA combines both TDA and BUA together. Suppose that we have a table T_0 and two privacy thresholds $(l-i+1)\delta$ and $(l-i)\delta$, where $1 < i \leq l$. From T_0, FLA directly creates two tables T_1 and T_2 with maximal utilities, where $P(T_1) < (l-i)\delta < (l-i+1)\delta$ and $P(T_2) < (l-i+1)\delta$. Based on T_2, FLA further builds up another table T_3 such that $P(T_3) < (l-i)\delta$, as shown by Fig. 6.4. Next, FLA compares the utilities between

$2 \times U(T_1)$ and $U(T_2)+U(T_3)$. If the former is greater, then FLA will publish T_1 under the privacy requirement as $(l-i+1)\delta$; otherwise, it will publish T_2.

When creating a table T_i with privacy level δ_h, FLA adopts TDA Phase I to create server master equivalent classes, and within each master class, FLA compares the utility loss by using BUA or TDA Phase II. Since BUA is more flexible for satisfying the privacy requirement, FLA adopts TDA Phase I to partition data into multiple sub-tables, and within each sub-table, FLA uses BUA and TDA to create equivalent classes. The final equivalent classes come from the partitions with maximal utility.

During TDA Phase I, we strictly use binary partitioning, as shown by Fig. 6.5. The reason is that at a lower privacy level, one may create several clusters, for instance, assuming 3 clusters, and then, at a higher privacy level, the number of clusters becomes less, assuming 2 clusters. Clearly, one cluster in the lower privacy level has to split and merge with others in order to achieve a higher privacy level later. Many data utilities will be lost while splitting an equivalent class, especially when the size of the equivalent class is large. Usually, the higher the privacy level is, the bigger an equivalent class is. Although during a table's propagation, tuples are clustering from lower privacy levels to higher levels, during the top-down splitting process, the partitioning results at the earlier phases satisfy the higher-level requirements, and then with the reducing of their sizes, they can no long meet the higher-level requirements.

For achieving a single privacy requirement δ_h, FLA first gradually partitions the table into several blocks such that each block meets the current privacy level. For example, in Fig. 6.5, data sets S_5 to S_{10} represent these blocks. Next, within their parent blocks, S_1, S_3, and S_4, FLA uses BUA to create equivalent classes. Finally, FLA compares the results of both BUA and TDA Phases, and adopts the best one as the equivalent class.

ID	Weight	Age	Zip Code	Drug
1	187	24	09210	Heroin
2	186	24	09211	No
3	186	23	09210	No
4	185	25	09212	Marijuana
5	185	24	09211	Marijuana
6	184	27	09212	Marijuana
7	184	25	09210	Heroin
8	184	26	09211	No
9	183	28	09210	Marijuana
10	181	28	09212	Marijuana
11	181	29	19222	Heroin
12	185	30	09212	No
13	185	31	09212	No
14	186	31	09212	No
15	174	28	19202	Marijuana
16	193	29	19212	Marijuana
17	189	30	19202	Marijuana
18	183	34	09212	Marijuana
19	176	22	19222	Heroin
20	185	21	19122	No
21	184	22	19222	No
22	195	24	19122	Marijuana
23	189	23	19123	Marijuana
24	192	24	19125	Marijuana

(a) original table

ID	Weight	Age	Zip Code	Drug
1	[187-195]	[23-29]	*92**	Heroin
2	[187-195]	[23-29]	*92**	No
3	[187-195]	[23-29]	*92**	No
16	[187-195]	[23-29]	*92**	Marijuana
22	[187-195]	[23-29]	*92**	Marijuana
24	[187-195]	[23-29]	*92**	Marijuana
5	[184-189]	[23-31]	*92**	Marijuana
7	[184-189]	[23-31]	*92**	Heroin
13	[184-189]	[23-31]	*92**	No
14	[184-189]	[23-31]	*92**	No
17	[184-189]	[23-31]	*92**	Marijuana
23	[184-189]	[23-31]	*92**	Marijuana
4	[176-185]	[22-34]	*92**	Marijuana
6	[176-185]	[22-34]	*92**	Marijuana
8	[176-185]	[22-34]	*92**	No
18	[176-185]	[22-34]	*92**	Marijuana
19	[176-185]	[22-34]	*92**	Heroin
21	[176-185]	[22-34]	*92**	No
9	[174-185]	[21-30]	*92**	Marijuana
10	[174-185]	[21-30]	*92**	Marijuana
11	[174-185]	[21-30]	*92**	Heroin
12	[174-185]	[21-30]	*92**	No
15	[174-185]	[21-30]	*92**	Marijuana
20	[174-185]	[21-30]	*92**	No

(b) FLA (no weights)

Figure 6.6: Example (Part I). Part II is in Fig. 6.7.

6.5.4 Example

Take Figs. 6.6 and 6.7 as an example. Fig. 6.6 (a) is the original table, which consists of 24 tuples. The attributes:"Weight", "Age", and "Zip Code" are the quasi-identifiers, the combination of which may unique identify a person. "Drug" is the sensitive attribute, which has three types of value: "Heroin", "Marijuana", and "No". The occurrence frequencies of these three types of value follows "Heroin":"No":"Marijuana"$= 1 : 2 : 3$.

Our proposed algorithms not only work on the condition that each attribute has a unique weight (also known as source-based utility weights) but can also be applied to the scenario in which tuples have personalized weights. In Fig. 6.6 (a), assume that there is a background knowledge in which tuples 1 to 10 are Hollywood stars, and tuples 11 to 24 are vagrants. Consider the fact that Hollywood stars care a lot about their weights and some of them use drugs, such as heroin, to lose weight. The "Weight" attribute is important for these groups of data, but it is not for the vagrants. Instead, the living area is an important factor in drug using habits. Therefore, during the creation of equivalent classes, one may give different weights to different types of users' data. In this example, we give 0.5 utility weight to "Weight" and another 0.5 weight to "Age" for the Hollywood group, while assigning 0.5 utility weight to both "Zip Code" and "Age" attributes for the vagrant group. Here, we let the table be propagated twice, and let $\delta = 0.05$. So, after the first time of transmission, the privacy disclosure of the result table should be smaller than $2\delta = 0.1$, and after the second time, it should be less than 0.05. Fig. 6.6 (b) and Fig. 6.7(a) show the sanitation results with and without using the personalized weights. Since the personalized weights put similar users' tuples closer regarding the overall attribute distance, Fig. 6.7(b) has more data utility than Fig. 6.6(b). Note that when using the personal weights the attribute distance is no longer symmetric.

When using FLA preparing for the first time of propagation, FLA creates two clusters: tuples $1-10$ and $11-24$. Since their privacy disclosure amount is 0.0069 and 0.0038, which are both smaller than the privacy requirement, FLA continues splitting the clusters into tuple groups $1-5$, $6-10$, $11-18$, and $19-24$. The four groups' privacy disclosure amount is 0.0292, 0.667, 0.0122, and 0 (as shown by Fig. 6.7(b)). Since any further partitioning breaks the privacy restriction, Fig. 6.7(b) is the result of directly using TDA with privacy requirement as 0.1. However, FLA also considers the utility cost of future sanitization, and therefore, FLA computes Figs. 6.7(a) and (c). In Fig. 6.7(a), FLA directly adopts TDA to create a table for the future privacy level $P < 0.05$, and based on Fig. 6.7(b), FLA builds a table

ID	Weight	Age	Zip Code	Drug
1	[184-187]	[23-27]	0921*	Heroin
2	[184-187]	[23-27]	0921*	No
3	[184-187]	[23-27]	0921*	No
4	[184-187]	[23-27]	0921*	Marijuana
5	[184-187]	[23-27]	0921*	Marijuana
6	[184-187]	[23-27]	0921*	Marijuana
7	[181-184]	[25-28]	0921*	Heroin
8	[181-184]	[25-28]	0921*	No
9	[181-184]	[25-28]	0921*	Marijuana
10	[181-184]	[25-28]	0921*	Marijuana
11	[174-193]	[28-34]	*92*2	Heroin
12	[174-193]	[28-34]	*92*2	No
13	[174-193]	[28-34]	*92*2	No
14	[174-193]	[28-34]	*92*2	No
15	[174-193]	[28-34]	*92*2	Marijuana
16	[174-193]	[28-34]	*92*2	Marijuana
17	[174-193]	[28-34]	*92*2	Marijuana
18	[174-193]	[28-34]	*92*2	Marijuana
19	[176-195]	[21-24]	19*2*	Heroin
20	[176-195]	[21-24]	19*2*	No
21	[176-195]	[21-24]	19*2*	No
22	[176-195]	[21-24]	19*2*	Marijuana
23	[176-195]	[21-24]	19*2*	Marijuana
24	[176-195]	[21-24]	19*2*	Marijuana

(a) FLA (P<0.05)

ID	Weight	Age	Zip Code	Drug
1	[185-187]	[23-25]	0921*	Heroin
2	[185-187]	[23-25]	0921*	No
3	[185-187]	[23-25]	0921*	No
4	[185-187]	[23-25]	0921*	Marijuana
5	[185-187]	[23-25]	0921*	Marijuana
6	[181-184]	[25-28]	0921*	Marijuana
7	[181-184]	[25-28]	0921*	Heroin
8	[181-184]	[25-28]	0921*	No
9	[181-184]	[25-28]	0921*	Marijuana
10	[181-184]	[25-28]	0921*	Marijuana
…	…	..	..	..

(b) FLA (P<0.10)

ID	Weight	Age	Zip Code	Drug
1	[181-187]	[23-28]	0921*	Heroin
2	[181-187]	[23-28]	0921*	No
3	[181-187]	[23-28]	0921*	No
4	[181-187]	[23-28]	0921*	Marijuana
5	[181-187]	[23-28]	0921*	Marijuana
6	[181-187]	[23-28]	0921*	Marijuana
7	[181-184]	[25-28]	0921*	Heroin
8	[181-184]	[25-28]	0921*	No
9	[181-184]	[25-28]	0921*	Marijuana
10	[181-184]	[25-28]	0921*	Marijuana
…	…	..	..	..

(c) FLA (P<0.05; based on above Fig.(b))

Figure 6.7: Example (Part II). The value of tuples $11-24$ in both Figs. (b) and (c) are the same as them in Fig. (a).

for the next privacy-level, as shown in Fig. 6.7(b). After computing the utility loss of the results, FLA finds out that by directly using Fig. 6.7(a) under the privacy requirements, $P < 0.1$ and $P < 0.05$ can save slightly more data utility than using Fig. 6.7(b) and (c). FLA takes Fig. 6.7(a) as final result.

6.6 Security Analysis

In this chapter, we consider an honest but curious attacking model, in which each participant of the system could be an attacker. More specifically, in this model, the attackers will always follow the rule for providing correct information to the next participant on a social chain. However, the attackers may be willing to learn more information than he supposes to do. Note that, in practice, the information propagation paths form a graph, instead of a chain, and therefore, the information that an attack is able to obtain only comes from his direct neighbors. In this chapter, a system is privacy-preserving if, for any participant, the overall information that he could learn from all of his neighbors is no greater than the information that he directly obtained from the neighbor, who has the closest distance to the source node v_0. For the ease of description, let $\|u-v\|$ be the social distance between users u and v, which is counted as the number of hops along the shortest path between them, $\|u-v\| = \|v-u\|$. Let T_{v_i} be the table possessed by user v_i and $H(A_n|T_{v_i})$ be the conditional entropy that one learns about the sensitive attribute based on table T_{v_i}.

Theorem 6.10. *Assume v_i, v_j are two nodes along the same information propagation path from v_o. If all participants adopt the same δ and attribute weights $\{W\}$, and $\|v_i - v_o\| < \|v_j - v_o\|$, then v_j's table must be a sanitization result of v_i's table.*

Proof. In the proposed scheme, along the information propagation path, the participants gradually hide more and more information by putting tuple values into a broader range. The participant at downstream takes the output from his previous participant as its input during sanitization operations. Since sanitization is irreversible, there exists a series of participants, $v_{i+1}, v_{i+2}, \ldots, v_{i+(j-i-1)}$ such that any table belonging to them is the sanitization result of its previous table. Therefore, v_j's table must also be a sanitization result of v_i's table. □

Theorem 6.11. *If all participants adopt the same δ and attribute weights $\{W\}$, for any pair of participants v_i and v_j with $\|v_i - v_o\| = \|v_j - v_o\|$, we have $T_i = T_j$ by using FLA.*

Proof. Each step in FLA is determined: for the same input table T and privacy requirement δ, the output of FLA is always the same. Note that, when using FLA, even if two tuples are identical, FLA uses the row numbers to provide a unique operation order to them. As a result, there is no stochastic event in FLA. Now, consider two shortest information propagation paths, assuming $\{v_1, v_2, \ldots, v_l\}$ and $\{v'_1, v'_2, \ldots, v'_l\}$. Since all participants use the same δ and the original data owner v_0 gives the same table to the first receiver on each chain, the outputs of v_1 and v'_1 are the same. By induction, we know that whenever $\|v_i - v_o\| = \|v'_i - v_o\|$, the sanitization results on v_i and v'_i will be the same. Therefore, for the same initial owner v_0 and parameters, the solution of FLA only relates with the shortest hop numbers from its executor to v_0. $\square$

Definition 6.12. A scheme is called anti-colluding if the total information gain from the tables of the colluding users, $H(A_n|T_{v_1}, T_{v_2}, \ldots)$, is not greater than cahoot's maximum information gain $\max_i H(A_n|T_{v_i})$.

Note that colluding attackers may not be direct neighbors with each other. Let T_{v_i} be the table that an attacker obtained from his social chain. Tables T_{v_i} and T_{v_j} may be created on different social chains.

Theorem 6.13. *For a given initial table T_0 at v_0, FLA is naturally against colluding attacks, when all participants adopt the same δ and attribute weights $\{W\}$.*

Proof. For a given threshold δ and propagation length l, the amount of disclosed information is only related to the recipient's distance toward v_0. According to Theorem 6.10, if the distance between v and the source is smaller than that of v', then the obtained information of v' is a subset of that of v. For a group of cahoots, the maximum information that they could obtain must come from the member with the least distance toward the source v_0. As a result, our scheme is against colluding attacks. $\square$

6.7 Evaluation

In this section, we conduct extensive real data-based simulations to evaluate the performances of our scheme. For the ease of comparison, we use black bars to represent the results by using the Bottom-Up Approach (BUA in Chapter 6.5.1); We use gray shadowed bars to represent the results of the Top-Down Approach (TDA in Chapter 6.5.2). The white shadowed bars stand for the results of the Forward Looking Approach (FLA in Chapter 6.5.3), which compares the following utility wastes of the best and the second best merging options.

6.7.1 Simulation Setup and Evaluation Metric

In the simulation, we use the Statlog (German Credit Data) Data Set, which consists of $1,000$ instances and 25 attributes. For the ease of simulation, we use the file named "german.data-numeric", which coded categorical attributes as integers. We let the first column be the sensitive attribute, and let the remaining 24 attributes be the quasi-identifiers. During simulation, we consider the weighted Euclidean distance between tuples as their merging costs:

$$d(\sigma_i(T), \sigma_j(T)) = \sqrt{\sum_{k=1}^{n-1} w_k \times (\sigma_i(\pi_k(T)) - \sigma_j(\pi_k(T)))^2}$$

where w_i is the attribute weight, $\sum w_i = 1$. In practice, this distance could be computed as Hamming code distance or the number of modified digits by using suppression. After creating an equivalent class, the values of its members' quasi-identifiers will be replaced with their mean value within the class.

Instead of directly computing the remaining utility of each table, we count the percentage of data utility that has been wasted after sanitization. Let $\sigma_i(T')$ be the members from the same equivalent class after a sanitization operation, and $\sigma_i(T)$ be

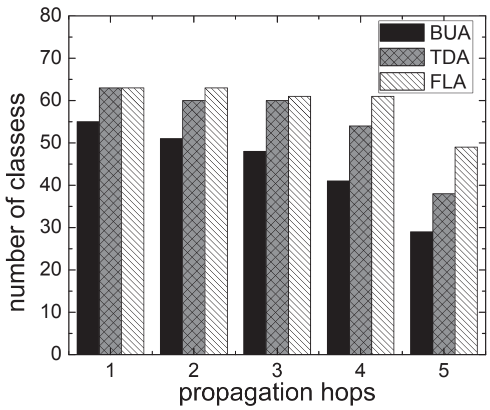


Figure 6.8: Created equivalent classes.

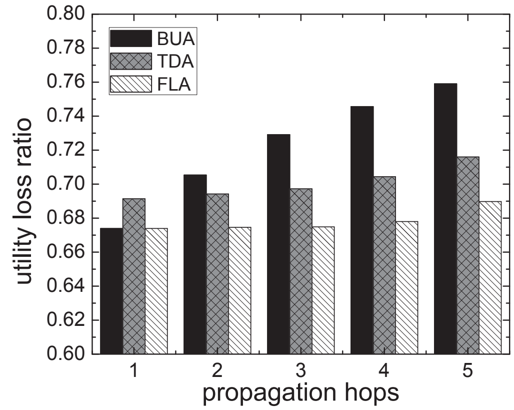


Figure 6.9: Ave. utility loss.

their corresponding original data, then the utility loss UL is defined as following:

$$UL(\sigma_i(T), \sigma_i(T')) = 1 - \frac{d(\sigma_i(T'), \sigma_i(\widetilde{T}))}{d(\sigma_i(T), \sigma_i(\widetilde{T}))}$$

where $\widetilde{T}$ stands for the resulting table by sanitizing the initial table T_0 into one equivalent class, $\widetilde{T} = \pi(T_0)$.

6.7.2 Simulation Results

Fig.6.8 shows the changing pattern of the number of created equivalent classes at each participating nodes during the table's propagation. Note that, the closer a participating node is to the source, the greater the corresponding table's utility that is preserved. As Fig.6.8 shows, the number of equivalent classes decreases along the information propagation path. However, the reducing speed is not linear. As we have mentioned, the maximum number of equivalent class is bounded by the number of sensitive values that have the minimum occurrences. Therefore, in the first hop, TDA and FLA create the same number of equivalent classes. During the propagation, more and more tuples merge into a larger group and form a new equivalent class.

Since certain approaches may inappropriately put some tuples into an equivalent class at an earlier time, the creation of equivalent classes with higher privacy-preserving requirements becomes harder and harder. As a result, they may build less equivalent classes than others at the later time.

The system's overall utility loss is shown by Fig. 6.9. In this figure, the x-axis indicates the total number of propagated hops, and the y-axis gives the average utility loss per table. From the figure, we can see that the average amount of utility loss increases with the growing length of propagations. FLA has the smallest increasing speed, while BUA has the largest one. In general, FLA preserves more data utilities than the other two approaches.

During the table propagation, although all result tables satisfy the same set of privacy requirements, the exact privacy-preserving values are different. Fig. 6.10 shows the changing pattern of each tables' privacy values. The horizontal lines show the increasing privacy requirements. The y-axis represents the privacy level of each created table along a propagation path, and we measure the privacy level by function $P(T)$ from Chapter 6.3.2.1. In most cases, BUA preserves more privacy than TDA. Since FLA considers both the current and future privacy requirements, at hop 3 in Fig. 6.10, it directly uses the privacy requirement of the fourth hop.

Fig. 6.11 gives the pattern of utility loss with an increase in privacy-disclosure threshold δ. Note that, in the previous three simulations, gradually create a serial of tables with increasing privacy-preserving levels: the input at hop i is the output of hop $i-1$. However, in Fig. 6.11, we always use the original table T_0 as input and assign different privacy-preserving level δ. From the figure, we can see that the amount of utility loss dramatically increases at the higher privacy-preserving threshold (a smaller δ). As we expected, the tables created by BUA loses the most data utility, since its tuple-merging scheme (at an earlier phase) may cause inappropriate overall

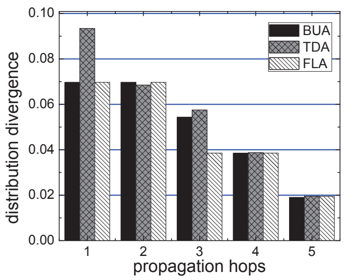


Figure 6.10: Ave. privacy.

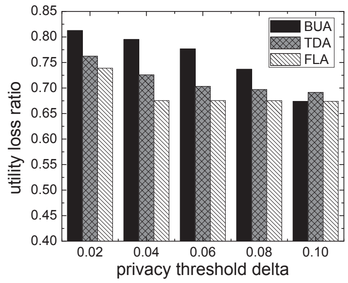


Figure 6.11: Utility lost.

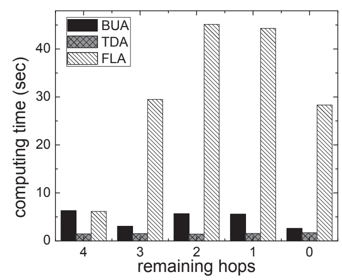


Figure 6.12: Time cost.

clustering results. We can also find that, at the condition of lower privacy-preserving thresholds, there are no significant differences between the TDA and FLA. The main reason for this phenomenon is that the members of each class created by these two approaches are similar to each other at the lower thresholds.

Finally, we consider the computing complexities of these three methods in Fig. 6.12. Note that the solution space of the attribute-level approach is in $O(n^2)$ while that of the tuple-level approaches is in $O(m^2)$, where m is the number of records, n is the attributes, and $m >> n$. As a result, the computing speed of the attribute-level approach is much faster than the methods at a tuple-level. Since the looking forward method uses both bottom-up and top-down approaches multiple times and creates several tables at different privacy levels, it takes a significant time to find tuples' merging results.

6.8 Chapter Summary

In the Chapter 5 of this dissertation, we have shown how to privacy-preserved publish time-evolving social-related data from a centralized system to others. In this chapter, we further study the privacy-preserving data publishing in distributed systems.

To our best knowledge, all of the existing studies focus on providing a single level privacy protection via a centralized scheme. Consider that information's recipients are not equally trustable, especially in a distributed social system, such as mobile social networks. In this chapter, we propose a new research problem: given a source node and a social path with length l, how are we to sequentially generate l data tables, such that their overall utility is maximized and data privacy is gradually enhanced along the propagation path. For solving the problem, one has to consider the similarity between individual records, together with the sensitive values' distribution within each equivalent class. Clearly, the problem is more complex than the conventional optimization problems of K-anonymity or t-closeness. In this chapter, we first design two local algorithms for optimally sanitizing tables with a single privacy-requirement, and then, we combine these two algorithms together and use a privacy requirement forward-looking scheme: during sanitization, instead of minimizing the utility loss under the current privacy level, our scheme considers the sanitization impacts on the future. Extensive simulation results show that our proposed algorithms can successfully increase the overall data utility, and meet the enhancing privacy-preserving requirements.

The contributions of our chapter are threefold.

1. To our best knowledge, we are the first to consider the multiple-level privacy-preserving social data publishing problem within a distributed system.

2. Since there are two objects with the problem, we propose a bottom-up approach and a top-down one, which give different priorities to the objects. In order to achieve maximal overall utility, we introduce the concept of 'forward looking': the sanitization at an earlier phase should not significantly jeopardize the data utilities on the later phase.

3. Extensive experiments demonstrate that the forward-looking scheme can produce a 5.6% higher total utility than the top-down and bottom-up

approaches.

CHAPTER 7

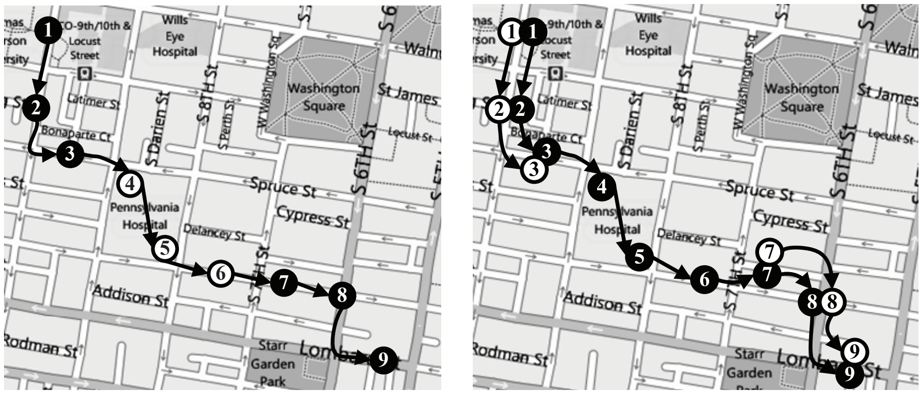
FRIENDSHIP-BASED LOCATION PRIVACY DISCLOSURE IN MOBILE SOCIAL NETWORKS

From data accessing aspect, malicious users can obtain any social information that a regular user may have. This chapter shows that the social information can be used for launching attacks. We consider a specific type of location-based service (LBS) known as a mobile social network (MSN), where users' social information is public. We demonstrate a new type of attack, where an adversary can combine the location and friendship information found in an MSN, to violate user privacy. We propose a fake location reporting solution that does not require any additional trusted third party deployment. We use extensive simulations to determine the validity of our scheme.

7.1 Introduction

The increasing popularity of smartphones has led to the rise of mobile social networks (MSN). An MSN is a combination of online social networks and location based services. An MSN can provide many new services, such as friends locator.

In order to provide such services, the MSN provider has to collect the location information from users and their friends. This has led to concerns that such information may pose a privacy threat since users may be unaware that they have



(a) single users (b) two close friends

Figure 7.1: An example of friendship-based location privacy disclosure.

revealed some sensitive information until after the fact. One apparent technique to protect privacy is to give users more control over when their locations are updated. The intuition is that individuals are the best arbiters of what locations are private, and by allowing a user to *not* upload his location at such sensitive locations, location privacy is achieved.

However, this intuition may not adequately provide location privacy against an adversary that knows both the location information, as well as social relationships. To illustrate, consider the map shown in Fig. 7.1, where the center region is a hospital. The number in each cell indicates the observation sequence. In Fig. 7.1 (a), a user may decide that the hospital is a sensitive area, and choose not to upload his location information when he is near the hospital (i.e. locations $4-6$). However, the adversary can use trace data collected over time to determine the positions, where the user did update his locations, and use that information to infer that the user did visit the hospital. The adversary can also use the user's friendship information to better refine the location prediction algorithm. In Fig. 7.1 (b), we see that the user does not upload his location near the hospital, but his friend continues to do so. Since the adversary is aware of the friendship information, the adversary can use the friend's location information to fill in the location gaps of that user. In this chapter, we consider the problem of providing location privacy against an adversary having access to the data collected by the MSN provider.

7.2 Literature Review

Anonymity can be provided via the frequent changing of pseudonyms [119, 120, 121, 122] such to make it difficult for adversaries to detect a user's movement. The system first defines several special regions named "mix-zones". In these fixed areas, a number of nodes enter the zone and another amount of nodes leave the zone. Within each mix-zone, nodes change their pseudonyms at the same time and do not report their locations in the zones. Because the transitions among nodes are simultaneous, the adversary cannot get the complete trajectories of users so that the mix zones preserve the nodes' location privacy.

However, for MSN-type applications, such as friend locator applications, the MSN provider will still have to know each pseudonym's friends, so as to deliver the correct information. As such, the frequent changing of user IDs is unlikely to mask the user's identity. Another technique of providing anonymity is to use k-anonymity techniques [123, 124, 125, 126]. In such techniques, a user will only upload the location of a region, which contains $k-1$ neighbors. This will ensure that the adversary cannot pinpoint the specific user. The use of k-anonymity in an MSN is not feasible because the user's friends can only receive an approximate area, and cannot determine the location of the user. This will make an MSN less useful.

The other category of location privacy preserving techniques is obfuscation. For this category of techniques, the user will try to confuse the adversary through techniques, such as injective noise [127, 128, 129], reporting fewer locations [130], reporting false locations [127] or increasing the intervals between reported locations [131]. Our approach follows this category.

The difference is that we consider a more powerful adversary who can utilize the *friendship* information to predict user movements better. Friendship information can be combined with location information to predict user movements.

7.3 Overview

Our system model consists of an MSN provider and many members. The adversary may control an MSN provider. Thus, the adversary has access to all of the location updates, and can use this information to filter out fake locations.

We divide time into discrete time units. A user's request can be generated at any time, but will only be transmitted to the MSN at a predefined time step. If there are multiple requests at a time step, the phone will only provide the nearest location. In order to protect users' privacy, at each time step, users can report either their real locations, fake ones or none. We assume that friends will be able to distinguish between a real or fake location, for instance via some unique IDs. Table 7.1 contains the notations used.

Table 7.1: Table of notation

K	Kalman gain (updating ratio)
H	Observation model matrix
F	State transition model matrix
I	Identity matrix
Δd	Extra distance provided by a user at a time step
$I(L_A; L_B)$	Mutual information between two sets of locations

7.3.0.1 Kalman Filter

The adversary will use the Kalman filter to estimate users' unreported locations. The Kalman filter is a set of mathematical equations that provide efficient computational (recursive) means to estimate the state of a process in a way that minimizes the mean of the squared error [64]. Forward and backward Kalman filling [64] is a technique, derived from the Kalman filter, to estimate the missing data of a linear system. It uses the Kalman filer's estimated states to represent the missing data. The Kalman filling algorithm first uses some reasonable data to pre-fill the missing data at corresponding time steps. Then, it applies the Kalman filter to the data and obtains a set of estimated states of every time step. Finally, it replaces the pre-filled data by these estimations. *Forward/backward Kalman filling* uses the data in the ascending/descending order of time.

Algorithm 18 Tracking algorithm based on Kalman filling

1: Input: H_R reported locations at each time step, H_E external knowledge for pre-fill the unreported ones in H_R
2: Use H_E to pre-fill the empty items in H_R, obtained H_F
3: Apply forward Kalman filter to H_F, obtained H_{KF}
4: Apply backward Kalman filter to H_F, obtained H_{KB}
5: **for** Each unreported location in H_R **do**
6: Replace it by $\alpha * H_{KF} + \beta * H_{KB}$ (α, β: weighted values)
7: Return Kalman filling result H_R

7.3.0.2 Adversary Location Estimation

Algorithm 18 illustrates how an adversary estimates a user's location. Since the trajectories of humans are continuous, and the moving pattern of people can be modeled by a linear process with the noise, the unreported locations in a single user's trajectory can be estimated by using forward and backward Kalman filling.

The adversary will first provide some synthetic locations to pre-fill the locations set, based on some outside knowledge. For example, the missing data can be linearly interpolated. Then, the interpolated data will later be used as measurements for the Kalman filter.

The adversary can use social relationships to predict a location as follows: the adversary can first determine a distance R_S. If the distance between user A and his friend, user B, is less than R_S, the adversary can regard them as being together. For all MSN friends of A, the adversary can calculate the percentage of being together with A. When A applies an unreported location in a protected trajectory, the adversary can guess that A stays with one of his friends. The partial traces of a user's friends can be used as an external location source to estimate the hidden locations during the use of Kalman filling.

7.3.0.3 Location Privacy Preserving Metric

The evaluation metric is the average error between guessing results and real locations at each reporting time. An adversary's error degree can be calculated as $W_{adv} = \frac{\sum_T \|Loc_G, Loc_R\|}{\sum amount(T)}$, where W_{adv} represents the average mistake degree of an adversary's guessing; $\| \cdot \|$ represents the Euclidean distance between a guessing location Loc_G, and its corresponding actual location Loc_R; $amount(T)$ represents the total amount of reported locations. If there is no fake or unreported location in the observations, the value of W_{adv} is 0. The higher the adversary error degrees are, the safer users' location privacy is.

7.4 Proposed Solution

7.4.1 User Traveling Alone

Here, the user is traveling alone. When choosing fake locations to update, he wants to select locations that cannot be easily filtered out by the adversary.

7.4.1.1 Relationship Between Estimation Error and Fake Location Distance

Suppose that in order to protect the privacy of his own trace, a user wants to use N% locations as fake locations. We assume that the user uses fake location $(x + \Delta x, y + \Delta y)$ at time k, where x and y represent the real location. Δx and Δy can be thought as user specified *noise*. Now, we want to calculate the relation between fake locations and their corresponding estimated locations by Kalman filter. The estimation error of the Kalman filter can be evaluated by a covariance matrix $P_{k|k}$, which stores the covariance of the state at time k based on observation at k.

$$P_{k|k} = (I - P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1}H)P_{k|k-1}, \tag{7.4-1}$$

where I is an identity matrix, $P_{k|k-1}$ is the covariance of the state at time k based on the past observation at time $k-1$, H is an observing matrix, R is the covariance of noise, k is a time instance and H^T is the transposed matrix of H. If the covariance of *noise* in a user's historical location set is relatively larger, the covariance between observation and prediction will be greater. In other words, the further a fake location is from the real location, the greater the estimation error will be.

7.4.1.2 Estimation Error of Kalman Filter

The Kalman gain K determines the trust between a predicting system to its observation and its predicting result [64]. Under conditions where R is constant, both the estimation error covariance $P_{k|k}$ and K will stabilize quickly and remain constant [64].

Now, we consider the total estimation error led by *noise* in a 1-D space. Assume that we add Δd to one location in x or y direction. Then, the additional estimation error Δe in the following n^{th} step can be represented as follows:

$$\Delta e = ((I - KH)F)^n K \Delta d, n = 0, 1, 2, ..., \tag{7.4-2}$$

where I is an identity matrix, K is the Kalman gain, H is an observing matrix and F is the state transition model. The trace of a user can be divided into x and y directions, which means the moving trace of a user can be seen as the combination of two traces from the 1-D space.

The Kalman filter can deal with noise because the expected value of noise is zero. As a result, if we provide *noise* in the same direction to a real location, the influence of fake locations can be accumulated and disturb the estimated result of the Kalman filter. However, the value of Δe will decrease exponentially when n goes up: If we provide a fixed value Δd to a group of traces, the extra estimation error is a fixed value: $\sum_n ((I - KH)F)^n K \Delta d$. We use $\lambda \Delta d$ to represent this fixed value. Suppose that we use N% of our reported locations as the fakes. Since a fake location can be considered as a real location adding Δd_i, the average estimation error of a trace is $N\% \times \lambda \overline{\Delta d_i}$.

7.4.1.3 Fake Locations Reporting Pattern

There are two options when reporting fake locations: 1. Concentratively report fake locations in some time, then continually report the real ones; 2. Provide the fake locations separately. The estimation error of Kalman filter at a place is the sum of errors caused by previous fake locations. However, as the parameter decreases rapidly, an intensive reporting of fake locations will cause the remainder of the locations to be less protected. Therefore, we will separately report fake locations to protect the whole trace. A user will only provide intensive fake location updates when protecting a specific sensitive location, not the entire trace. Hence, if a user wants to protect a certain location, he can intensively report some fake locations. As for the condition of using fake locations to distort the whole trajectory, it is better to report the fake locations separately.

7.4.1.4 The Optimal Fake Location

Users' moving trajectories are a group of points in a two-dimensional space. To find the best fake locations at each point means to find the maximum value of $\Delta x^2 + \Delta y^2$ with speed restrictions. Suppose that (x_0, y_0), (x_1, y_1) and (x_2, y_2) are three consecutive points in a trajectory. Assume that (x, y) represent the corresponding fake location to point (x_1, y_1). With the limitation of speed, the maximum distance a user can reach in the time interval is R. The fake location (x, y) should satisfy the following:

$$R = S_{max} * \Delta t; \tag{7.4-3}$$

$$(x - x_0)^2 + (y - y_0)^2 \leq R^2; (x - x_2)^2 + (y - y_2)^2 \leq R^2; \tag{7.4-4}$$

$$Goal : maximize(\sqrt{(x - x_1)^2 + (y - y_1)^2}) \tag{7.4-5}$$

Normally, a user makes a U-turn far less than others. In order to reduce the complexity of the algorithm, we always use the intersection point of two speed limitation circles as the best fake location. Hence, the best fake location (x, y) can be computed as follows:

$$a = (\frac{y_2 - y_0}{x_2 - x_0})^2 + 1 \tag{7.4-6}$$

$$q = \frac{(x_2 - x_0)(x_2 + x_0) + (y_2 - y_0)(y_2 + y_0)}{2(x_2 - x_0)} - x_0 \tag{7.4-7}$$

$$b = -2(y_0 + \frac{y_2 - y_0}{x_2 - x_0} q), c = y_0^2 - R^2 + q^2 \tag{7.4-8}$$

$$(x, y) = (q + x_0 - \frac{y_1(y_2 - y_0)}{x_2 - x_0}, \frac{-b + \sqrt{b^2 - 4ac}}{2a}), \tag{7.4-9}$$

$$or(x,y) = (q + x_0 - \frac{y_2(y_2 - y_0)}{x_2 - x_0}, \frac{-b - \sqrt{b^2 - 4ac}}{2a}) \tag{7.4-10}$$

However, the best fake locations are obtained based on the assumption of knowledge of the next location. We assume that there is an application that can predict the next position (x_2, y_2) by using a current actual location and a current instantaneous speed, or users may provide the next position intended for hiding current sensitive locations.

Among all of the optimal fake locations of a trace, we note that the fake locations, which bring more errors to Kalman filling, are always the turning points' corresponding fake locations. In our method, we let each user's application record the distribution of the distances between each of the real locations and their corresponding fake ones. Therefore, reporting N% fake locations means finding a distance threshold D such that $\sum Dist(d \geq D) = N\%$. At each time interval T, if the distance between a fake and real location is greater than D, the application will report this fake location, otherwise it will report the real location.

Algorithm 19 Fake location generation algorithm

1: Input: Previous location (x_0, y_0), current location(x_1, y_1), current speed S_C, time interval T, maximum speed S_{max}
2: Use (x_1, y_1) and S_C to predict next location (x_2, y_2)
3: Compute the two potential fake locations based on formula 9-10, and calculate the distances from them to (x_1, y_1)
4: Return the furthest potential fake location (x, y)

Algorithm 20 Single user trace protection

1: Input: Real location (x_1, y_1), fake reporting percentage N, fake distance distribution $Dist$ and candidate fake location (x, y)
2: Compute the distance Δd between (x_1, y_1) and (x, y)
3: **if** $\sum_d Dist(d > \Delta d) \leq N$ **then**
4: Return the fake location (x, y)
5: **else**
6: Return the current location (x_1, y_1)

7.4.2 Users Traveling in a Group

The probability of two users appearing together is proportional to the closeness of them. We use the distance distribution between two users (A and B) to measure the closeness of them. We apply the concept of mutual information to analyze the problem. The historical location sets of A and B can be represented by two random variables L_A and L_B. We use $I(L_A; L_B)$ to represent the mutual information between A and B's locations sets. A high amount of mutual information between two users means that they always appear in the same regions at the same time.

The closeness of two users is subjective. We use a predefined threshold to determine the state of *being together*. We first compute the distance's distribution of two users. Then, we calculate the probability that two users appear in the same region at the same short period of time, based on a predefined threshold. The probability represents the chance that the two users stay together.

According to Information Theory, mutual information also represents how much the uncertainty in guessing L_B has been removed by knowing L_A. If user A reports his location at time t while B doesn't, the uncertainty of those unreported locations of B will be reduced since adversaries can get more information from A's locations. According to the definition of mutual information, by increasing the uncertainty of getting together, or by increasing the probability of arriving at some location alone, the location binding mutual information between users will become less.

A high amount of mutual information between A and B also means a great value of $Prob(L_B|L_A)$. The adversary can guess the position of B at a certain time t by $Prob(L_B) = Prob(L_B|L_A) \times Prob(L_A)$. If we can increase the uncertainty of A's reported locations, then B's locations can be protected better, and vice versa.

Our fake location generating method works as follows: when the distance between A and his friend, B, is less than a predefined *being together* distance threshold, the two users will report their real location and fake location at the same time, in turns. For example, A reports his real location at time t_1 and B reports a fake location. Then, at time t_2, B reports the actual location and A reports the fake one. Through this method, we reduce the mutual information between the two users and also increase the uncertainty of reported locations.

7.5 Evaluation

To perform the simulations, we established a Cartesian coordinate system where the point $(0, 0)$ is the beginning of a user's trajectory. We set the initial user speed to be 1 unit of distance per unit of time, with a maximum speed of 7 units. At each step, we provide a normally distributed noise as the state noise and another normally distributed noise as observing noise. We use the state noise to represent the speed and moving direction change of a user. Both observation noise and state noise are used to randomize the movement change pattern of a user. The average speed of a user is about 3.5 units of distance per time. In experiments, we vary observing time, continuous unreported locations and the closeness between two friends.

7.5.0.1 Single User Results

We compare our scheme against the *location omission* strategy (the user will pick $N\%$ locations and not report them), as shown in Fig. 7.3. The figure shows the original trajectory of a user and the trajectory estimation results before and after using our fake locations-based method. Fig. 7.3 is a part of a random trajectory consisting of 2,000 points. We select 10% as a fake location reporting ratio. Fig. 7.3 shows that the Kalman filling result is inaccurate by our choice of fake locations.

For evaluating the effectiveness of our method to different random trajectories, we conducted the following simulation: we fixed the maximum speed and the

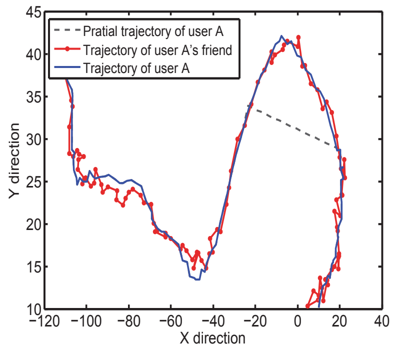


Figure 7.2: Two users travel together.

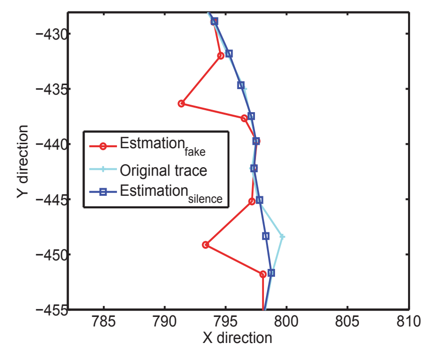


Figure 7.3: One user defense.

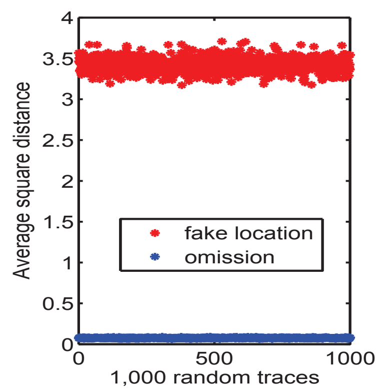


Figure 7.4: Error by fake or omission.

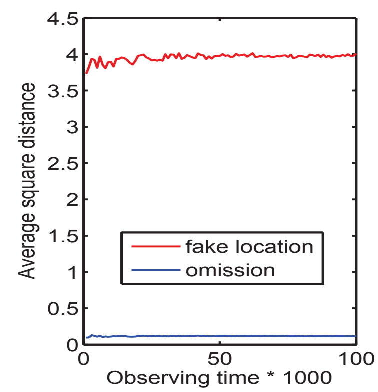


Figure 7.5: Error vs. observing time.

initial position, left the other parameters randomized and generated 1,000 different trajectories. In this experiment, each trajectory was observed 2,000 times. The average speed of these traces varies from 2.73 to 5.39 units of distance per time. Fig. 7.4 shows the average estimation error per observation time between protecting via fake locations and omission, and we can see that our method can significantly improve the traces' privacy of users.

Next, we want to show that our method is always effective, regardless of the length of adversaries' observations. We generate 1,000 random traces, and the observing times of each trace change from 100 to 10^5. The fake reporting ratio and the maximum speed are the same as the previous setting. Fig 7.5 shows the average error's change pattern as time goes on: the average distance approaches a constant number.

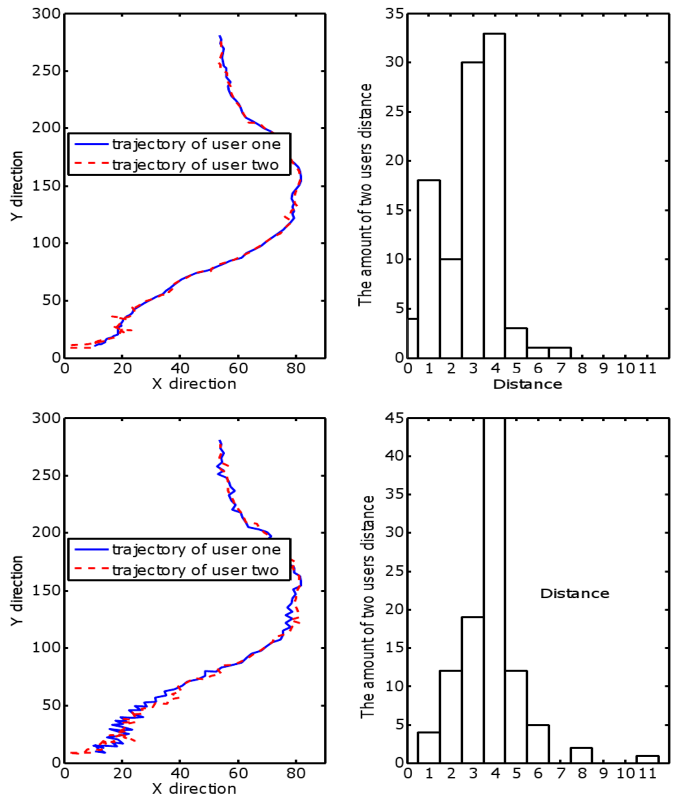


Figure 7.6: The trajectories and distance histogram between two friends before and after using our proposed method.

7.5.0.2 Two Users Results

The adversary may compute the closeness between two users using a statistic closeness metric. When the distance between two users is less than a threshold, we regard them as being together. If the closeness of two users is $k\%$, then when one user does not report his locations, there is $k\%$ probability that he is near his friend.

In Fig. 7.7, we let the number of continuous unreported locations vary from 1 time step to 50 time steps. We compute the average estimation error among 1,000 random trajectories, and to each trajectory, we observed 1,000 steps. We found that when users do not report their locations in a short period of time, Kalman filling has better estimation results, while, if users do not report their locations for a long time, using the position of users' friends has a better estimation result.

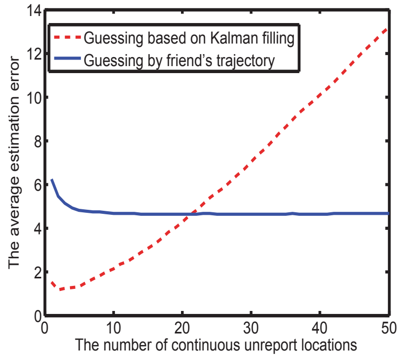


Figure 7.7: Error vs. unreported num.

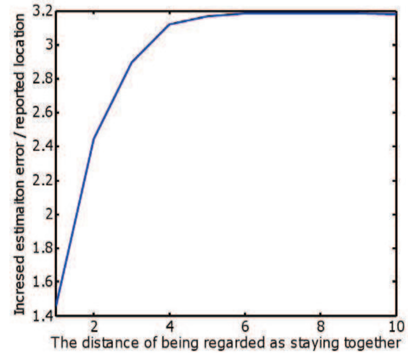


Figure 7.8: Error vs. threshold.

Next, we illustrate the results before and after using our two users privacy-preserving method. Fig. 7.6 upper left shows the original trajectories of two users, and the upper right picture is the histogram of their distance. If we defined the distance between them as less than 4 units of distance apart, then 64% of their locations are satisfied. However, after using our method, the percentage reduces to 38%.

In order to see the relation between our method and the closeness threshold, we tested 1,000 pairs of users. Each pair was observed 1,000 times. The average speed of these tested trajectories varies from 2.82 to 3.76 units of distance per time. The maximum speed is set to 5 units of distance per time. Fig. 7.8 shows the change pattern of average estimation error when closeness-distance standard changes from 1 to 10 units of distance. As the adversary increases the threshold, his guessing result will depend more on the locations of the user's friends, since the closeness of the users will be increased, leading to the estimation error increase.

7.6 Chapter Summary

Although using social information can improve the performance of certain applications, malicious users can also use social information to launch new attacks. In this chapter, we consider the social information-based location privacy in MSNs. We introduce a new privacy attack where the adversary uses both historical movements and friendship information to estimate a user's hidden trajectory. In order to protect user's location privacy, our solution allows a user to upload fake locations so that attackers are not able to correctly calculate the closeness between friends.

The contributions of the chapter can be summarized as follows:

1. To our best knowledge, we are the first to consider the social information-based location privacy-disclosure attack.
2. We analyze the problem by information theory and provide a scheme to determine the best fake location.
3. Our solution does not require the use of trusted third parties, which may be difficult to deploy in the real world, to provide location privacy protection.

CHAPTER 8

SYBIL ATTACKS IN MOBILE SOCIAL NETWORKS

Using social information does not mean blind trust. Before incorporating any information in a system's decision-making process, one has to check the authentication or trustworthy first. Chapters 8 and 9 give two attacks caused by the lack of sufficient verification mechanism.

In this chapter, Chapter 8, we study the Sybil attacks. By creating a large number of fake identities and their friendships, malicious users can gain a disproportionately high benefit through a Byzantine fashion. Actually, the applications introduced in Chapters 2, 3, and 4 are all vulnerable to Sybil attacks. In order to limit the effect of Sybil attacks, in this chapter, we propose a local ranking system for estimating trust level between users.

8.1 Introduction

In recent years, researchers have designed different frameworks for cooperative services of smartphones [48]. However, the open environment makes these systems vulnerable to *Sybil attacks*. In a Sybil attack, an adversary creates a large number of fake identities (Sybils), and since all Sybils are controlled by the adversary, she can subvert the system by making actions that benefit herself. Here is an example of a data sharing service. In order to reduce a cellphone's usage of cellular networks, when several nearby users are watching the same online content, they locally share partial data (which has been downloaded to each other's phone) via other networks; the amount of shared data a user obtained depends on the quantity he provided. The attacker can benefit from the service by only downloading one portion of data and obtaining other parts via different Sybil identities.

Traditional social network-based Sybil defenses are based on two common assumptions: (1) Sybil-free social networks are fast-mixing; (2) Sybils can only fool a limited number of honest users. Intuitively, a fast-mixing network consists of a single well-connected core, even after the decomposition of the graph into its k cores [132]. However, these assumptions lead to two problems. First, the accuracy of a Sybil defense algorithm is highly related to the number of attack edges. Second, it is unclear whether the assumption about the fast-mixing feature will hold in all social networks. Research by [132, 133] suggested that in some social networks, there are multiple cores with considerable sizes. Since we cannot be sure that fast-mixing will always be present, it is reasonable to make the following assumption: *honest users may cluster into one community or several communities with similar sizes.* But, under the new assumption, using most existing Sybil defenses will result in high false positive rates (many honest users from other honest communities will be labeled as Sybils).

In order to solve the problems, we design a Sybil defense system in mobile social networks. Considering that trust is pairwise instead of global, our system explores both trust and distrust relations among users. For addressing the problem of multiple honest communities, our system uses a distributed Sybil defense algorithm based on signed networks. For reducing the impacts of attack edges, we propose a gateway-breaking algorithm. Note that the connectivity between honest communities is different from that between communities of an honest and a Sybil. *If we cut off several high centrality edges from the social graph, the connectivity between honest nodes bears much less of an impact than that between Sybil and honest nodes.* This algorithm *potentially* increases the accuracy of any social network-based Sybil defense algorithm.

8.2 Literature Review

Neighborhood Monitoring-based Sybil Defenses: since attackers have a limited number of real devices, a group of Sybils is actually sharing one device. Based on this observation, Sybils can be detected by letting honest users monitor signals' features [134, 135] or moving patterns of other users [136]. Paper [137] proposed a strategy for detecting Sybils in mobile networks: each user locally stores a friendship graph and a foe graph. Whether a suspected node will be regarded as a Sybil is dependent on the similarity of the graphs between the involved users. But, their solution is not suitable for the late participants or frequent travelers. Moreover, none of the above schemes work well if the attackers only conduct malicious actions to a small set of honest users instead of all users.

Social Network-based Sybil Defenses: based on the small-cut structure of a social network where a Sybil node is more likely to be connected with Sybil nodes and a Sybil can only fool a limited number of honest nodes, graph theorems are usually adopted to detect Sybils [138, 139, 76]. Our proposed scheme is also based on the unique link structures of Sybil nodes, but unlike previous works, we use a more realistic model that the honest users present single or multiple communities. Under this model, most of the existing social network-based Sybil defenses suffer high false positive rates.

Signed social network [140] is a new type of social networks. Unlike traditional social networks, signed networks allow users to add both trust (positive) and distrust (negative) relations to their social profiles [141]. In our system, we *create* a signed social network behind a given mobile social network to capture the mutual relations among users. The idea of our system is that, by using the created social graph, Sybil nodes can be detected through the propagation of both trust and distrust.

8.3 System Model

Our system consists of two parts: a remote server and users. The server is responsible for two jobs: (1) storing and periodically pruning the created signed network graph; (2) assigning *randomly sampled* social profiles to users for computing the trust level between users.

We assume that each honest user has one mobile phone, which is associated with a single identity, while attackers may associate multiple fake identities with one device. For the remainder of the chapter, we call the identities held by the attackers as *Sybil identities*, and we also refer to identities as nodes. Each identity is required to periodically send a special message to the server to keep valid, and the server will return updated social profiles. Unlike traditional models, we assume that the honest region of a social network may form several communities. Exactly how many honest communities may be formed is determined by the social networks being considered.

Our system works as follows. Each user locally stores two *randomly sampled* social profiles: a trust and a distrust profile. Whenever two strangers encounter and want to establish cooperative service, each user's phone will exchange the trust profile (together with signature and timestamp), and locally compute a trust and a distrust score to determine whether the other user is a Sybil. In order to increase the accuracy, a special pruning algorithm is running on the server.

In our system, to capture the trustworthiness between users, we *create* a signed network. If node u trusts (distrusts) node v, then $E^+_{uv} = 1$ ($E^-_{uv} = 1$). The trust relations come from the friendships of users, so E^+ are always bidirectional, while the distrust edges are unilateral. $N^+(v)$ indicates a list of directly trusted

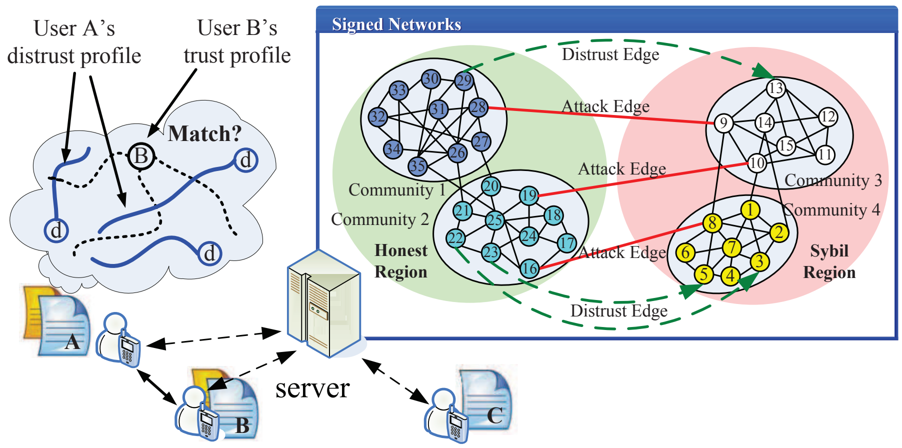


Figure 8.1: System model.

nodes of v, and $N^-(v)$ gives the directly distrusted list. For example, in Fig. 8.1, $N^+(v_6) = \{v_5, v_7, v_8\}$, and $N^-(v_{29}) = \{v_{13}\}$.

8.4 Signed Network-based Sybil Defense

8.4.1 Generation of Signed Network

Our system detects Sybils by exploring both trust and distrust relations. But, how to find the distrust relations is an open question. Here, we propose several options. First, consider that multiple Sybils are sharing a single phone and that each Sybil identity needs to report some message periodically in order to keep itself valid. For some honest users, they may catch the instant that an attacker switches her Sybil identities. If that is the case, then the honest users will report this misbehavior to our server, and a distrust edge will be added from the reporter to the accused. Second, when several honest users, who have been fooled by the same *attacker* via different identities, physically encounter the attacker at the same time, some of them may notice that the attacker is using a different identity; the honest one could report this event to our server. Besides these two options, any other neighbor monitoring techniques may also be adopted.

Our system does not require that each honest user must monitor other users. Instead, we just use the data provided by some cooperative users, who are willing to report the misbehavior events they witnessed. Since the attackers can use hit-and-run policy, sharing the knowledge about the abnormality increases the accuracy of our system. Traditional reputation system-based solutions only maintain a global trust value for each user while a signed social network preserves much more information, especially when adversaries have a target list while act normally to other nodes.

8.4.2 Scheme of Signed Network-based Sybil Defense

When user A needs to interact with a nearby stranger B, both of them will locally measure a trust level between each other by using *Signed Network-based Sybil Defense algorithm* (SNSD), as shown by Algorithm 21. SNSD makes use of social profiles, which are stored on users' phones. SNSD first computes a trust score and a distrust score. Based on the scores, it further classifies nodes into: *trusted*, *neutral*, or *distrusted*. Because the Sybil-free social networks may contain several communities, we use the neutral tag to label the honest users from other honest communities. Similar to most social-based Sybil defense algorithms, SNSD also applies random walks to represent the propagation of trust and distrust. Since the random paths are generated by the server, the attackers cannot control how the random walks are conducted.

8.4.2.1 The Generation of Social Profiles

Each user locally records a trust and a distrust social profile. When a new user V joins our system and provides his trust-friend list, the server will generate the two profiles. The generating procedure for a trust-relation profile is as follows: the server first sends out K random walkers from V. Each walker will conduct an l-length

Algorithm 21 SNSD Algorithm (Run on node V)

Exchange trust social profile with the other user S.
Trusted degree calculation:
Extract paths from the trust social profiles of V and S.
Set $Ver(V, S)$ to be 0.
for Each suspect path P_s **do**
  if P_s intersects more than half of verifier paths **then**
    $Ver(V, S) = Ver(V, S) + 1$.
Compute trusted degree by Equation 1.
Distrusted degree calculation:
Extract paths from the distrust social profiles of V.
for Each suspect path P_s **do**
  if P_s comes across more than half of distrusted verifier paths **then**
    $Dis(V, S) = Dis(V, S) + 1$.
Compute trusted degree by Equation 2.
According to Equation 3, label the suspect S.

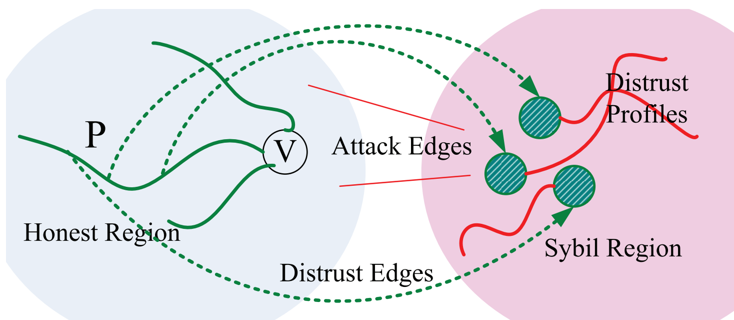


Figure 8.2: The generation of distrust social profiles.

random walk along trust edges. A generated path represents one possible way of trust propagation. These paths will be sent to V as a trust social profile. Obviously, the profile is a random sample of V's l-hops friendship.

In order to impersonate real users, attackers have to create a large friend set for each Sybil identity. Usually, such friendships are set up by letting Sybil nodes friend each other. Therefore, friends of a distrusted node are likely to be distrustful. Our server generates V's distrust social profile by using the distrust relations of both V and his trusted friends. Take Fig. 8.2 as an example. First, a distrust seed set is generated: along trust edges, the server computes $K = 3$ short-length random paths from V (solid green lines). The ends (shadowed circles), which are distrusted by the

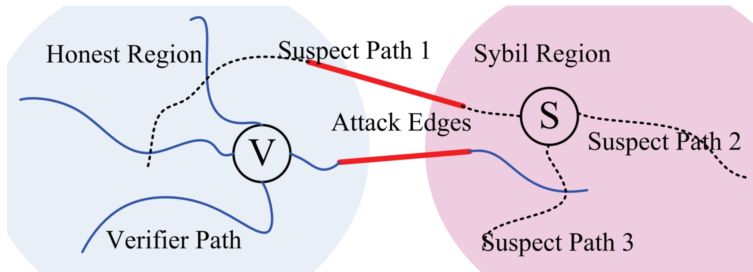


Figure 8.3: The computation of trust score.

nodes on these paths, build up the seed set. From each seed, another l-length random walk will be conducted, and the paths (solid red lines) will be used as the distrust social profile of V.

8.4.2.2 Trust Level Estimation

The computation of the trust-level is based on the similarities between users' social profiles; SNSD gives a probability value to represent to what degree a node can be trusted. Whenever strangers V and S encounter, they will exchange their trust-social profiles. After obtaining the profiles, V will locally compute a trust score and a distrust score for S. For the ease of description, the paths in V's trust social profile are named as *verifier paths*, and the paths in the trust profile of S are called *suspect paths*, as shown by Fig. 8.3. The solid blue lines indicate V's trust social profile, and the black dashed lines represent the trust social profile of S. If there is a common node on both a verifier path and a suspect path, the suspect path is verified once; when a suspect path has been verified more than k_t times, where k_t is a constant, we say that this suspect path is fully verified. Let $Ver(V, S)$ be the number of fully verified paths, and recall that there is a total of K random paths in a trust social profile. In regard to V, the trust score of S is given by: $Trust(V, S) = Ver(V, S)/K$. In Fig. 8.3, when V sets the verifier threshold $k_t = 2$, then only suspect path 1 will be fully verified. $Ver(V, S) = 1$, $|K| = 3$, and $Trust(V, S) = 1/3$.

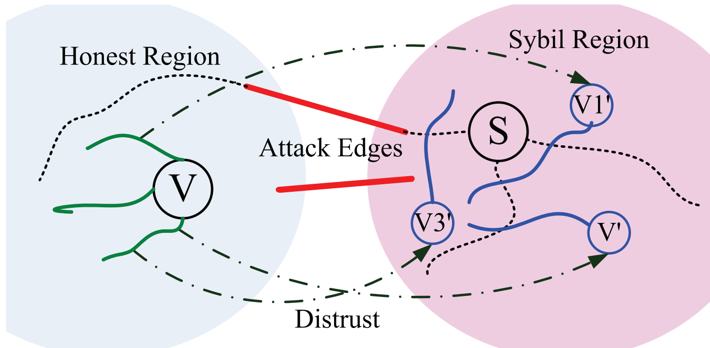


Figure 8.4: The computation of distrust score.

For the computation of distrust score, SNSD considers V's distrust social profile and the trust social profile of S, as shown in Fig. 8.4. We name the paths from V's distrust social profile as *distrust verifier paths* and use K' to represent the size of V's distrust social profile. When there are k_t distrust verifier paths having common nodes with a suspect path, this suspect path is a fully verified distrust path. Let $Dis(V, S)$ be the total number of fully verified distrust paths, and the distrust score of S in regard to V is given by: $DisTru(V, S) = Dis(V, S)/K'$. The distrust social profile of V contains 3 distrust paths (solid blue lines). In Fig. 8.4, if we define a verified distrust path as a suspect path that comes across at least half of the distrust verifier paths, then only suspect path 3 is verified. $Dis(V, S) = 1$, $K' = 3$ and $DisTru(V, S) = 1/3$.

The final label of S, $L(V, S)$, is determined by the difference of the two scores: $z = Trust(V, S) - DisTru(V, S)$. Let α, β be two thresholds, $1 \geq \alpha > \beta \geq -1$. When $z \geq \alpha$, then $L(V, S) =$'Trusted'; if $\alpha \geq z \geq \beta$, $L(V, S) =$'Neutral'; otherwise, $L(V, S) =$'Distrusted'.

8.4.3 Security Analysis

Sybil attacks are hard to defend against because Sybil nodes can impersonate honest nodes by supporting each other. However, SNSD forces the attacker into a dilemma. On the one hand, in order to boost the trust scores, it is better for Sybils to cluster into one community, such that the verifier paths are more likely to encounter a suspect path. As the attacker cannot predict how many and from which attack edges the trusted verifier paths may enter into the Sybil region, single community structure provides more chance for having contact with verifier paths. On the other hand, for reducing the distrust scores, the attacker should build Sybils into multiple communities for two reasons. (1) the probability of incurring the distrust verifier paths is proportional to the size of a Sybil community; having a single Sybil community may result in high distrust scores. (2) if a distrust verifier path enters Sybil region, having multiple communities can trap it in one of the communities instead of threatening all Sybils.

Although the attackers can adopt bad mouthing strategy, in our system, such strategy has very limited impacts on honest users. The main reason is that the distrust seeds come from an honest user's close friends. When the attackers create more distrust edges to honest users, the chance of having these fake distrust edges in other honest users' profiles is small, unless the attackers intensively add the fake distrust edges to a node. But if it is the case, our pruning algorithm will detect this high centrality structure. For attackers, the bad mouthing strategy brings many more problems than benefits.

8.5 Sybil Gateway Breaking

In order to improve the accuracy, a special algorithm called Sybil gateway-breaking is proposed. Essentially, the algorithm prunes some suspicious edges of the signed social network.

Algorithm 22 Sybil gateway-breaking algorithm (SGA)

Suspicious Edges Selection:
Select edges with high local betweenness as suspicious edges.
Use signed network-based Sybil defense algorithm (Chapter 8.4) to determine honest and Sybil.
Find shortest paths from the honest nodes to Sybil nodes.
Compute the visiting frequency of edges.
Take the edges with high visiting frequency as suspicious.
Gateway Verification:
Generate the initial neighbor set, $\{u\}$, $\{v\}$, and $\{w\}$.
Compute the number of unique paths from $\{v\}$ to $\{u\}$, and from $\{v\}$ to $\{w\}$
for Predefined times **do**
    Respectively add Δk disjoint neighbors into $\{u\}$, $\{v\}$, $\{w\}$.
    Compute the number of unique paths from $\{v\}$ to $\{u\}$, and from $\{v\}$ to $\{w\}$
    Compute the growing speeds of unique paths, S_{vu} and S_{vw}
    if $|S_{vu} - S_{vw}|$ is greater than a threshold **then**
        E^{+}_{uw} is a gateway;
E^{+}_{uw} is not a gateway.
Attack Edge Detection:
Find attack edges from detected gateways by distrust relations; break them.

8.5.1 Overview

The accuracy of most existing social network-based Sybil defense algorithms is related to the number of attack edges. If a malicious user can establish more attack edges, the overall accuracy of a Sybil defense will decrease; if the attacker concentratedly establishes attack edges to a targeted group of users, there will be more victims. But, if we can remove some attack edges, the accuracy can be increased.

Based on the above idea, we propose Sybil gateway-breaking algorithm (SGA), as shown by Algorithm 22. Consider that all of the paths connecting honest and Sybil nodes must go through the attack edges. The connectivity from a group of honest nodes (honest region) to a Sybil region is bounded by the number of attack edges. If each honest node can locally check whether it is fooled by others based on the connectivity, and deletes attack edges, the accuracy of *any* social network-based Sybil defense will be enhanced.

SGA consists of three parts: (1) Suspicious Edge Selection Algorithm; (2) Gateway

Verification Algorithm (GVA); and (3) Attack Edge Detection Algorithm. GVA is the core part of SGA. It determines whether an edge is a gateway, which connects different communities. Since there may be multiple honest communities, only part of the gateways are the attack edges while others are not. By exploring the distrust relations, the attack edge detection algorithm locates bad gateways. In order to reduce the computing time, a suspicious edges selection algorithm will be adopted at the beginning of SGA.

8.5.2 Gateway Verification Algorithm

Since GVA is the core of SGA, we will discuss it first. Whether two nodes locate in the same community can be verified by their connectivity to other nodes. Intuitively, if one node's connectivity to the third node is much larger than that of the other node, it is possible that the two nodes reside at different communities. Here, we use the number of unique paths to measure the connectivity feature.

Definition 8.1. *Unique paths indicate a group of paths connecting two distinct nodes or regions without sharing a common edge.*

Because the amount of unique paths connecting two nodes is bounded by node degrees, we compute the unique paths from a region to another region. Let u and w be the ends of a given edge, v be the third node for checking the connectivity, and $UP(u, w)$ represent the number of unique paths from u to w. Based on the community structure of these three nodes, we may observe one of the three cases: (1) Either u or w shares the same community as v. (2) u, w and v locate in the same community. (3) None of them comes from v's community.

For the first case, assuming that nodes u and v reside in the same community, $UP(v, u)$ is much greater than $UP(v, w)$ since all the paths from v to w must go through the gateways between communities, which are limited. If we gradually increase the size of regions by Δk, the number of unique paths from w's region will stop growing much earlier than u's region. But, for the other two cases, we will not observe such differences. Based on this, we design a gateway verification algorithm (GVA) for checking whether a given edge is a gateway. The procedure is as follows. GVA gradually adds Δk disjoint neighbors into both regions, which respectively contain u and v (or w and v), and examines the amount of unique paths between the regions. Since we only care whether the edge is a gateway, GVA only checks the existence of the growing speeds' difference for a given node v.

The selection of the third node v is based on the fact that most random walks are trapped in their initial community. Before checking whether E^{+}_{uw} is a gateway, GVA first sends out several l-length random walks from node u or w, and lets the end nodes be the members of the verifier set. During the computation of the growing speed, GVA sequentially selects nodes from the set to be the third node v.

However, checking every edge of a social network by GVA is impractical. For fastening the process, we need to find out a set of suspicious edges first. Based on this idea, we propose a suspicious edge selection approach.

8.5.3 Suspicious Edge Selection

There are two options for finding the suspicious edges: one is based on node centrality, and the other focuses on the edge centrality. The reason for caring about the node centrality is that under target attack, the target nodes will have high centrality values. Hence, the server can first generate a local map (G) of each node, and compute each node's centrality on G. The GVA is only applied to the edges, which are connected with the high centrality nodes. The criteria of centrality we used is called the betweenness centrality [142], which is defined as the number of shortest paths passing a node out of the total number of shortest paths within a given network. $B(w) = \sum_{u,v \in G, u \neq v} g_{uv}(w)/g_{uv}$, where g_{uv} is the total number of shortest indirect paths linking nodes u and v, and $g_{uv}(w)$ is the number of those indirect paths that include node w.

Since all of the paths between a Sybil node and an honest node must traverse the same set of attack edges, the suspicious edges could be the edges passed by a majority of random paths, which connect the nodes with antagonistic relations (at least one directly distrust edge between the nodes). Therefore, the server randomly selects several pairs of antagonistic nodes, creates random paths between each pair of antagonistic nodes and counts the visiting frequency of the transited edges. For the edges with high frequency, the gateway verification algorithm will be adopted. The above procedure substantially examines the centrality of edges based on partial nodes' relationships.

8.5.4 Attack Edge Detection

A gateway connects two communities together. Whether a gateway is an attack edge is determined by the distrust relationships between the communities. If either one of them, or both of them highly distrust the other, it is very likely that the gateway is an attack edge. Consider that the majority of random paths are trapped inside their own communities; instead of counting the number of distrust relations between two communities, we adopt random sampling to estimate it.

Attack edge detection algorithm works as follows: for each gateway, the server temporarily breaks it, and then, from its both ends, the server sends out k random walkers along the trust edges, respectively. The length of the random walks is a small fixed number, and all of the visited nodes build up a sampling set. The server also creates another set, called a distrust sampling set, which consists of nodes distrusted by the sampling set's members. The intersection of these two sets indicates the intensity of distrust of the communities. The larger the intersection set is, the more likely the gateway is an attack edge. The server permanently removes the gateways with high intensity.

8.6 Performance Analysis and Evaluation

Simulation setup: to generate a social network with different communities, we create 2 to 6 communities with 256 nodes in each; edges inside a community are randomly deployed according to the power law distribution. Links between different communities are also randomly generated, and 5% of the nodes from each community are connected to others. We compare our methods with the SybilGuard [139]. For the ease of description, we call our Signed Network-based Sybil Defense "SS", name our Sybil Gateway-breaking Algorithm "GB", and use "SG" to represent SybilGuard.

8.6.1 Simulation Metric

AFNR and AFPR: the first metric we applied is called Average False Positive Rate (AFPR) and Average False Negative Rate (AFNR). If an honest node falsely regards another honest node as Sybil, we call the event a false positive (FP). A false negative (FN) means a Sybil being regarded as an honest node. Since each honest node locally determines whether to accept other nodes, we compute the False Positive Rate (FPR) and False Negative Rate (FNR) at each node, and then, compute the averages of them. FPR = FP/(FP+TN), FNR = FN/(TP+FN), where TP represents the total amount of true positives, and TN represents that of true negatives.

SCR: A distributed network system can tolerate some faults made by Sybils. We term such robustness Sybil Conquered Rate (SCR). We assume that if more than 1/3 of the accepted nodes of an honest user are Sybils, the honest node is conquered by the attacker. SCR is defined as the number of defeated nodes out of the total amount of honest nodes. The smaller the SCR is, the better a Sybil defense algorithm is.

8.6.2 Simulation Results

Since it is possible that an honest region may consist of multiple communities, the number of honest communities may affect the accuracy of the Sybil gateway detection algorithm. We made one Sybil community and one (up to five) honest communities. The result is given by Fig. 8.5: with the number of the honest community increasing, the accuracy of our Sybil gateway detection algorithm also increases.

In order to reduce the computing time, we filter out a group of suspicious edges by their betweenness. From the light-colored bars of Fig. 8.6, we can see that the number of suspicion dramatically drops with the increase of betweenness threshold. But the majority of attack edges (Sybil edges) are still captured by the suspicion group, as shown by the dark-colored bars of Fig. 8.6.

Based on the results of the betweenness experiment, we produce a derived experiment: what could happen if we directly delete all of the suspicion edges without using our gateway-verification algorithm? We created an honest community and a Sybil community, and then, we remove the suspicion edges. SybilGuard is used to detect Sybils. The results show that directly deleting suspicions has a negative effect on accuracy, which also indicates the necessity of using our gateway-verification algorithm. One possible explanation is that the suspect edge group contains too many regular edges (about 90% in Fig. 8.6); removing these edges fundamentally changes the structure of the network, and it causes some honest nodes to become closer to the Sybil community.

Next, we focus on the accuracy of our proposed algorithms. We examined the impacts of the number of attack edges on SCR, as shown in Fig. 8.7. The more

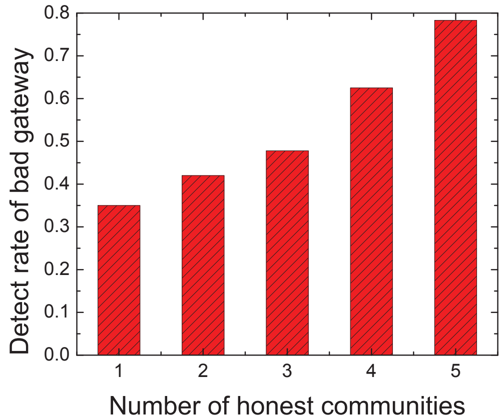


Figure 8.5: The impacts of the number of honest communities.

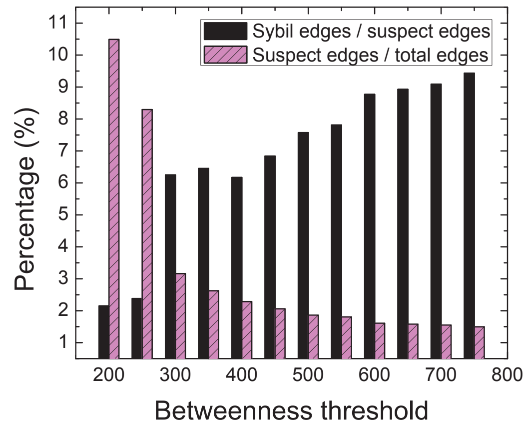


Figure 8.6: The impacts of the betweenness threshold.

attack edges are created, the more nodes are conquered. The result of SS+GB is better than that of SG, which means the performance of SG has increased after using our Sybil gateway-breaking algorithm. The SCR value of our accepted nodes (SS+GB) is about 4% less than that of SG. By further checking identities of rejected nodes, we find that our algorithm does eliminate some Sybil nodes near attack edges, which are accepted by SG.

Since SS is based on both trust and distrust relations, our next tested factor is the number of distrust edges, as shown in Fig. 8.8. Clearly, with the growing number of distrust edges, the accuracy of our algorithm will be increased. Moreover, by letting 20% nodes having one distrust edge, the SCR of SS is about 3.5% less than that of SG.

Our next consideration is the number of Sybil communities shown in Fig. 8.9. We kept one honest community, and gradually add more Sybil communities. For SS, there are approximate 4.75% distrust relations among all the relations. With the growth of Sybil communities, the AFNR of both algorithms increases in the beginning, and AFPR drops. However, when the number of Sybil communities becomes four and

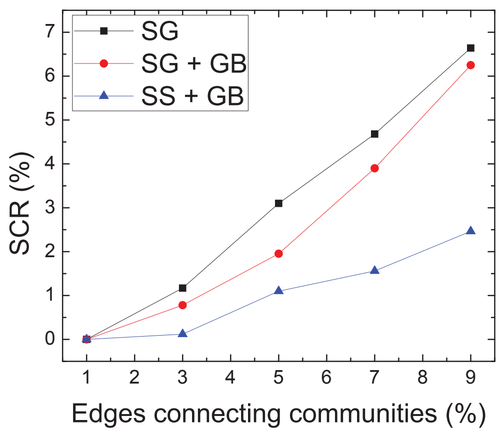


Figure 8.7: The impacts of the number of attack edges.

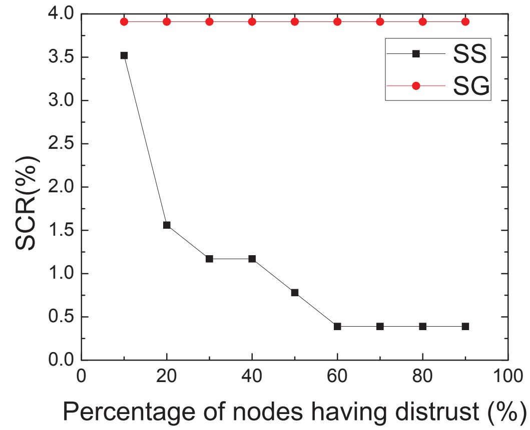
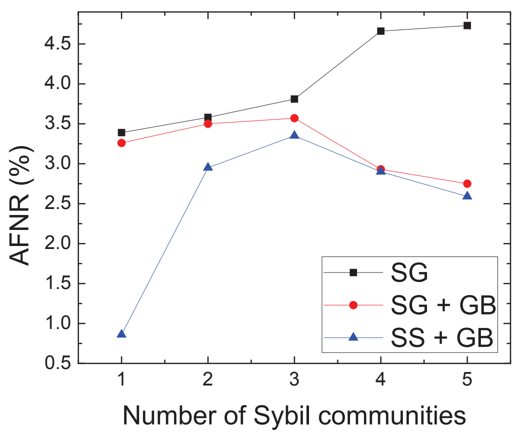
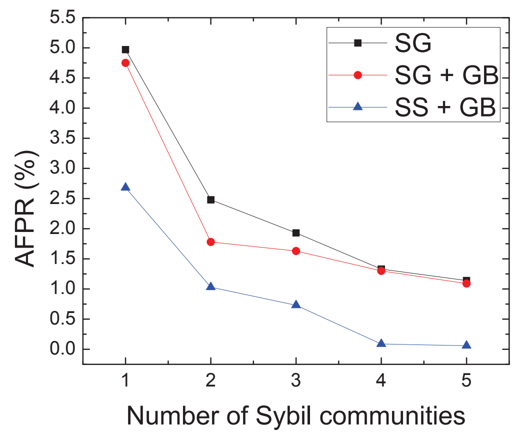


Figure 8.8: The impacts of the number of distrust relations.



(a) avg. false negative rate.

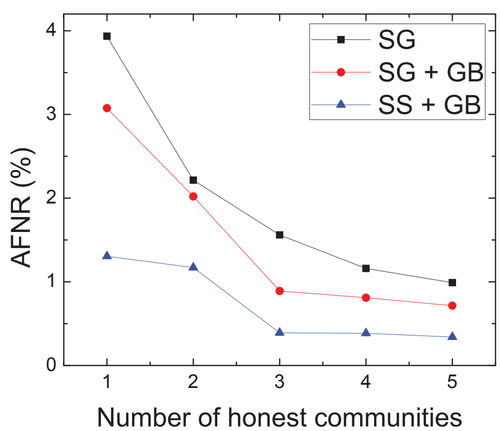


(b) avg. false positive rate.

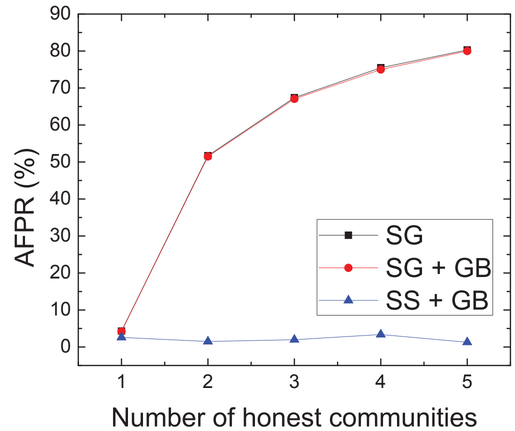
Figure 8.9: The impacts of the number of Sybil communities.

five, the AFNR values of SG+GB and SS+GB decrease. The reason is that with the growing number of Sybil communities, the AFNR values of both SG and SS increase, while the result of GB becomes more accurate. The integrated effects of them cause the AFNR to increase firstly and then decrease.

Our last consideration is the impacts of the number of honest communities on the accuracy of the three algorithms. In this part of the simulation, we have one Sybil community and one (up to five) honest communities, with 256 nodes in each one. The simulation results support our viewpoint that the existing Sybil defense algorithms



(a) avg. false negative rate.



(b) avg. false positive rate.

Figure 8.10: The impacts of the number of honest communities.

have high false positive rates when honest users cluster into multiple communities. From Fig. 8.10, we can see that both AFPR and AFNR of SS+GB is better than that of SG or SG+GB. Moreover, after adopting the gateway-breaking algorithm, the AFNR value of SS is decreased.

8.7 Chapter Summary

From the consideration of security, social information or its source must be verified before using it; otherwise, malicious users may take advantage of uncensored social information to launch attacks. In this chapter, we use Sybil attacks in mobile social networks as an example to show the necessity of social information verification. Unlike traditional Sybil-defenses, we use a more realistic model that the social network of honest users may not be fast-mixing. Based on this model, we propose a new Sybil defense algorithm by exploring both trust and distrust relations. Our solution is lightweight: users only need to carry two small social profiles instead of the whole graph. Our algorithm solves the problem of high false positive in the existing Sybil defense algorithms. For increasing the accuracy, we propose a gateway-breaking algorithm, which can be used by any social networks-based Sybil defenses.

The contributions of the chapter are as follows:

1. We propose a distributed verification mechanism to defend Sybil attacks in mobile social networks.

2. Our scheme explores both trust and distrust social relationships among participants.

3. We conduct extensive simulations to prove the significant performance of our approach.

CHAPTER 9

WORMHOLE ATTACKS IN COOPERATIVE TRAJECTORY MAP-BASED NAVIGATION

Some social information-assisted applications are built over the actual social relations among users. A typical example is a cooperative trajectory mapping-based friend locator application, which is introduced in Chapter 4 of this dissertation. However, there is an inexplicit assumption for using the cooperative trajectory mapping that all participators must be trustable. Due to the absence of data validation, once participators inject wrong information into such a system, the system's service cannot be guaranteed. In this chapter, we consider the problem of fake trail attacks in cooperative trajectory mapping. In a fake trail attack, the adversary seeks to create a fraudulent shorter path between two locations. The unique characteristics of cooperative mapping make conventional wormhole defenses, such as packet leashes, unsuitable. We propose an efficient algorithm which can successfully deter an adversary from launching such an attack, and we validate the effectiveness of our solution through extensive simulation experiments.

9.1 Introduction

The functionalities of smartphones continue to grow with many such phones now being equipped with hardware such as sensors and accelerometers. Cooperative trajectory mapping is an emerging technique that takes advantage of the capabilities of these smartphones to allow users to combine the data collected from their phones to create maps of user movements in an unknown indoor region. This type of map is known as a *cooperative trajectory map*. Trajectory maps are used in various applications such as people localization [49, 54, 55], public transportation tracking [51, 52, 53], and traffic monitoring [50].

Cooperative trajectory mapping typically does not use the phone's GPS in order to avoid the weak signal issue in indoor conditions. Instead, the smartphone's accelerometer and compass are used to determine the displacement and moving direction of a user [48]. This data is periodically transmitted to a central repository which collects and processes the readings from multiple users to arrive at a trajectory map. Prior work on cooperative trajectory mapping focuses on building the system and improving map accuracy and assumes that all participants are honest. However, the presence of malicious users can have a detrimental effect on the final trajectory map.

We can illustrate the effects of a malicious user in Fig. 9.1. We assume that the trajectory map is used to locate friends in an indoor scenario. Fig. 9.1(a) shows the reported trajectories of the three users (i.e. two honest users and one adversary). Fig. 9.1(b) shows the reported trajectories without the adversary. The routing results are given by Figs. 9.1 (c) and (d). The shadow areas represent rooms, and we assume that each room has only one door: people cannot go through the walls. In Fig. 9.1, user A is trying to locate user B. Using the collected data (Figs. 9.1(a) and (b)), the central depository computes the shortest path from A's current position to B (shown by the arrow lines in Figs. 9.1(c) and (d); the background map is not available to the server). Assuming that A is walking faster than B, A will eventually catch up with B. In Fig. 9.1(a), a malicious user, adv, reports a fake path. The depository may use this incorrect information and compute a different path for A. This path is incorrect since it requires A to go through the walls of a room, as shown in Fig. 9.1(c). Without adequate protection from malicious users, the resulting trajectory map will

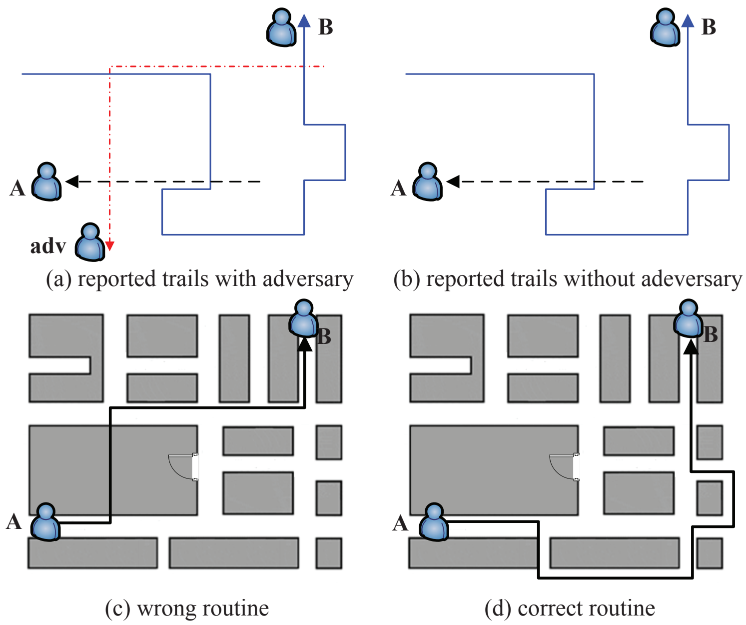


Figure 9.1: The effect of a wormhole attack.

be inaccurate, or even will become useless.

In this chapter, we name this type of attack as *fake trajectory attack* since the adversaries launch the attack by reporting some fake trajectories or fake encounters with other users. The fake trajectory attack is similar to a wormhole attack, considered in ad hoc networks [143] since we can consider the intersections as nodes and road segments as links of nodes. In a traditional wormhole attack, the adversary will maintain an out-of-band connection between two physical locations in the network. Using this out-of-band connection, the adversary creates a wormhole that can let nodes that are far away from each other appear to be neighbors [144]. As a result, the routing algorithm in ad hoc networks will report an incorrect path between two locations. In our cooperative trajectory mapping problem, an adversary that reports a fake path between two locations will create a wormhole between those two locations. The effect is the same as the traditional wormhole attack since the central depository will use this incorrect information to derive incorrect paths for users.

However, the unique characteristics of cooperative mapping make it difficult to apply existing wormhole defenses.

Firstly, traditional wormhole detection mechanisms assume that every node's sensorial range is R. We can estimate the distance between any nodes based on the neighboring table. In our model, the length of each road segment may be not the same; the number of intersections is not a constant but will grow.

Secondly, conventional wormhole attacks are strongly related to the geometry of the network: most wormhole detection algorithms assume that partial information about a wormhole-free case is available. However, in cooperative trajectory mapping, the original map data is not available to the server.

Lastly, traditional *time and spatial leash*-based wormhole defense approaches let the data packages identify the wormhole; however, in our problem, we have people following the route rather than packets. Unlike packets which can be retransmitted easily, moving people to alternative routes is time-consuming.

In this chapter, a novel centralized algorithm is designed for the detection of fake trajectory attacks. Considering the uncertainty of a user's instant location, we first estimate the possible position of every user at time t based on the reported trajectory. After that, we determine whether two users will definitely have physical encounters, may have physical encounters, or will definitely not. Then, we compare our estimation results with the reported encounter records. If there is an inconsistency among the data, we will give some penalty to the users. After observing enough time, we use two thresholds to classify the users into three groups: "honest group", "adversary group" and "suspicious group", and differentially use the data from different groups.

9.2 Literature Review

The idea behind the cooperative trajectory mapping problem considered in this chapter was proposed by [48]. There, a mobile social network-based navigation system is designed. Each user periodically reports his trajectory and encounter information to the server, then the server can reply with a routing to the user in order to help the user find others. Other work by [57, 58] also used similar ideas for cooperative trajectory mapping. The main difference is that prior work only considers honest users. Our work focuses on *non-colluded*, dishonest users that will intentionally report incorrect paths to disrupt the system.

Detecting fake paths in cooperative trajectory mapping shares similarities with traditional wormhole detection in ad hoc networks. In a wormhole attack, the adversary will attempt to convince other nodes that a path exists between two locations when, in fact, there is no path between the nodes. Using the same methodology as [144], we divide wormhole detection techniques into centralized and decentralized solutions.

Centralized wormhole detections, such as statistic detection [145, 146, 147] and multi-dimensional scaling [148, 149], rely on the server knowing certain node distributions or network geometric layout information, and using this information to detect the wormhole. However, in our trajectory mapping problem, the server does not have any knowledge about the real map, except for the reported moving trails of users. In fact, the server builds up a map using the mobility paths of users. Thus, existing centralized wormhole detection techniques cannot be applied to our problem.

Decentralized wormhole detection techniques rely on the nodes within the network to monitor the other nodes or data transmissions to detect a wormhole. Work by [150, 151, 152] requires a node to monitor the topology structure of its neighbors, while [153] lets the node monitor the input and output traffic flows. Other techniques that rely on cryptographic monitoring include packet leashes [154], TESLA [155], and distance bounding [156]. The main difference is that, in our problem, an individual node-reported path may not be used until it can be associated with information from a fixed AP or the global uniform coordinate system. Thus, a node cannot monitor other nodes in our problem.

9.3 Background

In this section, we will first introduce the system model in our problem. After that, we will discuss how to build a trajectory map based on the system. Finally, the adversary model will be presented.

9.3.1 System Model

A cooperative trajectory mapping system has the following three basic components.

1. *A service provider.* The service provider deploys a server to collect user data and to use that information to build and prune the trajectory map. The provider also provides additional services based on the eventual map, such as a friend locator. Except the collected data, the server does not have any domain knowledge about the real map.

2. *Users.* Users report their trajectories and encounters to the server.

3. *Access point (AP).* An AP is an additional, optional component. It works as a fixed location reference, such as a WiFi access point, whose physical location is known. The AP will periodically broadcast beacons. The purpose of the AP is to quickly establish the spatial relationship between each user's local movement trajectories and the actual physical coordinates. If all of the reported

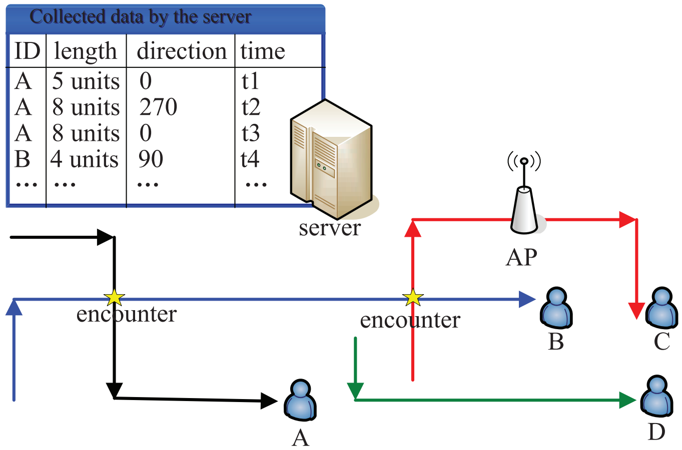


Figure 9.2: System model.

trajectories are noise-free and all of the trajectories are interconnected with each other, the additional AP is useless.

We assume that there are M users that participate in this system and that everyone's clock is loosely synchronized. Each user's mobile phone is equipped with an accelerometer, a compass, a wireless receiver, and an encounter sensor. The accelerometer and compass are used to determine that user's displacement and direction, respectively. The wireless receiver, such as a WiFi radio, is used to receive beacons transmitted from the AP. The encounter sensor is used to periodically signal the user's presence as well as record the presence of other users. This can be accomplished using a Bluetooth module built into the smartphone. For instance, when user A walks past user B, A's phone will record as "encountering B", while B's phone does the same. We use the unit disk graph (UDG) model to represent the sensorial range of an encounter sensor.

9.3.2 Trajectory Map Construction

A valid user will maintain two lists in his smartphone: *movement list* (ML) and *encounter list* (EL). The ML consists of a serial of displacement from the last recorded position. The tuples in ML are $\{senderID, displacement, direction, time\}$. The encounter information, which is the information about meeting with other users, is obtained by an encounter sensor. When a device learns about an encounter with another device, it will create an entry in the EL, as $\{senderID, time, encounter'sID\}$. The device of each user will periodically report these two lists to the server via a 3G connection.

However, the trajectories of each user are recorded in their own isolated coordinate system, which is only relative to the initial (unknown) location of the user [48]. If all of the trajectories are connected with each other (spatial and temporal encounters), the server can compute the relatively spatial relationships of intersections. Otherwise, we need a public location reference to provide unique spatial information about all of the users such that the spatial encounter near an AP can also be used as a real physical encounter. In a cooperative trajectory mapping system, an AP is randomly set at a place. When a user passes by the communication range of the AP, the user will also record the AP as an encounter. From the trajectories of the users, who have AP encounter records, the users' individual trajectories can be moved into a uniform coordinate system by joining the places of an AP encounter together. After A has passed by B, who has passed an AP, the trajectory of A will also join the uniform coordinate system since the spatial relation between A and the AP can be obtained by that of B and the AP, and A and B.

The server applies two variables to store the reported data: the first one is used to store the trajectories which have already joined the uniform coordinate system, and the second one is used for temporarily storing the space-relation unknown trajectories. Therefore, once a space-relation unknown user can reference to the AP, the server will move his historical trajectories from the unknown group to the other group.

One application of the trajectory map is a friend locator, which can be accomplished as follows: every user periodically reports his movement traces to the server, and the server stores the reported data. Periodically, the server will use the data to build or update a map by adding the new found paths to the map. Since the server records the current position of each user, the service provider can use the built map and the recorded location to find the shortest routing between users.

9.3.3 Adversary Model

The goal of an adversary is to launch a wormhole attack by creating a fake, shorter path between two locations. If the distance between the fake path and the real path is less than a system defined value R_e, then we consider it as an estimation error. We assume that the server has some other algorithm to deal with estimation errors. Thus, the adversary has to select a fake path that deviates from an actual path by at least R_e.

We assume that there are multiple adversaries, but they are all working independently. The number of real users is assumed to be larger than the number of adversaries. An adversary can create a fake path by manipulating the ML or the EL. The adversaries can modify or create any item in their current ML and report them to the server such that some shorter, fake paths will be included in the trajectory map. Adversaries can also modify or create any record in their current EL and report them to the server such that some real users' trajectories will be wrongly added in the map, which may also generate some shorter but nonexistent paths.

In this chapter, if the adversary reports a trajectory, which he did not follow, but the trajectory corresponds to an existing path on the map, we do not regard it as an attack. However, if the associated encounter information with this trajectory is wrong, we view it as a fake trajectory attack.

9.4 Detection Algorithm

In this section, we will present our adversary detection algorithm. The structure of this section is as follows: In the beginning, we will present the intuition behind our solution. Then, our adversaries detection and isolation (ADI) algorithm will be discussed in detail. The trajectory-based encounter-type prediction is a key function used in the ADI algorithm. We will also discuss this prediction in this section.

9.4.1 Intuition

A key feature of cooperative trajectory mapping is that when a user encounters another user, each user will independently report encountering the other user to the server. The intuition behind our algorithm is to make use of this feature to detect an adversary. We illustrate the intuition using the following example.

Consider an adversary that is physically at location α (loc_α) but reports to the server that he is at loc_β. Since the adversary is not at loc_β, he cannot determine whether there are any users at loc_β. Considering that there is a large number of users in real applications, the probability of guessing correctly is very small, and we do not consider this case. Now, let us assume that an honest user is at loc_β. Since the adversary is not at loc_β, the honest user does not report encountering the adversary. This results in an inconsistency, or an *abnormal event*, since a supposed encounter did not occur. The honest user hence acts like a *witness* that can be used to identify an adversary. Over time, an adversary will be associated with more abnormal events than honest users, and the server will use this as a means of identifying adversaries.

Based on the above idea, in this chapter, we present the adversaries detection and isolation (ADI) algorithm. In the ADI algorithm, we assign a suspected degree to each user. The suspected degree presents the percentage of abnormal events to the number of encounters. The algorithm periodically checks the intersections of each pair of users, estimating the possible *encounter-type* of the users. There are three types of encounters: two users cannot encounter, may encounter or must encounter with each other. Then, we use the estimated encounter-type to compare with the reported encounter information. By this way, we can find the abnormal cases: (i) must meet with no encounter, (ii) impossible meet with an encounter(s), and (iii) those with a single-sided encounter record. We update the suspected degree based on the detected abnormal events. After enough data is collected, we use pre-defined thresholds to find out the identities of the adversaries, suspicious and honest users.

The key challenges behind our algorithm are: (a) how to correctly determine abnormal events from a set of trajectories and encounters. Since the exact movement patterns of users are unknown and the server is only given the displacements and its corresponding time, there is uncertainty about the happening of encounters; (b) how to correctly assign a suspicious degree to the found abnormal cases since assigning wrong suspicious degree may cause an honest user to be regarded as an adversary.

9.4.2 Adversaries Detection and Isolation

When the server runs the Adversaries Detection and Isolation (ADI), as shown in Algorithm 23, it traverses all displacements and encounter records to find whether there are abnormal cases in which the trajectory information is inconsistent with the encounter records.

At the beginning of the ADI algorithm, each pair of users u and v is assigned with a suspected degree $SD(u,v)$.

$$SD(u,v) = \frac{SD.suspect}{SD.witness}$$

The initial value of $SD(u,v)$ is zero. The suspected degree consists of two parts. The first part (SD.suspect), the numerator, is a suspected penalty counter. The ADI

Algorithm 23 Adversaries detection and isolation algorithm (ADI)

SD.suspect= 0, SD.witness= a small non-zero number
for each time period T **do**
  if the result of TBEP (Algorithm 24) or encounter records satisfy the requirement of being a witness **then**
    SD.witness=SD.witness+1
  if the result of TBEP conflicts with encounter records **then**
    SD.suspect=SD.suspect+1
Find the suspects by threshold
Build trajectory map without using the data of suspects
Return: identities of suspects and the cooperative trajectories map

algorithm will check every users' reported displacements. Based on the displacements, we can estimate the possible encounter-type of the users. The trajectory-based encounter prediction algorithm (TBEP) will be presented later in this chapter. Once the algorithm detects an inconsistency between the encounter prediction result and the reported encounter information, the $SD(u, v)$ of the users will be given a penalty. Admittedly, an adversary with knowledge of the ADI algorithm might intentionally frame honest users by reporting false encounters. We will discuss some details in Chapter 9.7. The second part (SD.witness), the denominator of the suspected degree, records the times of being a witness: (i) the user's trajectory encounter with others' trajectories, or (ii) there is an encounter report about the user.

After observing the reported information for a long enough time, we can classify the users into three groups: an "honest group", a "suspicious group", and an "adversary group" by using two pre-defined cluster thresholds, threshold 1 and threshold 2 (threshold 1 > threshold 2). If the value of $\sum_v SD(u, v)$ is above threshold 1 or below threshold 2, the corresponding users belong to the "adversary group" or the "honest group", respectively. The remaining users fall into the "suspicious group", which needs to be further investigated (observing more time). In the phases of building the cooperative trajectory map, we first build the map only using the trajectories of the honest users. Then, we only partially and temporarily revise the map if there is no path between two users without the corresponding paths from the suspicious group'

Table 9.1: Table of notation for TBEP

t_B	beginning shared time of two displacements
t_E	end shared time of two displacements
$\{(x_i^A, y_i^A, t_i^A)\}$	a series of displacements of user A in T
S_{max}	maximum speed
EC	potential encounter conditions and time
d	length of a displacement
SF_H	head boundary function
SF_T	tail boundary function
ER	encounter records
R	sensing range
$PTB()$	possible trajectory boundary function
$IT()$	time-space relation function

trajectories. We do not use the data from the adversary group.

9.4.3 Trajectory-based Encounter-type Prediction

The function of the Trajectory-based Encounter-type Prediction (TBEP) algorithm is to estimate the encounter-type of any two given trajectories. In this part, we will first present the definition of some terms which we will use. Then, we will introduce the TBEP algorithm. Finally, some related theorems will be presented. The details of the help functions, which the TBEP algorithm calls, will be discussed in the next section.

We will illustrate TBEP by using the example in Fig. 9.3. There are two displacements of users A and B (Fig. 9.3(a)). We assume that they finished the displacements at the same time. If we can obtain the instant speed of the users at any time, we can clearly find out when and where those two users met with each other in the past. However, determining the exact position of each user at any point in time is difficult to obtain without knowing the instant speed. Knowing the instant speed of a user is difficult to obtain in a realistic scenario.

However, considering the maximum speed limitation, we still can find a user's feasible position bound at any time. The *feasible position bound* is a time-related

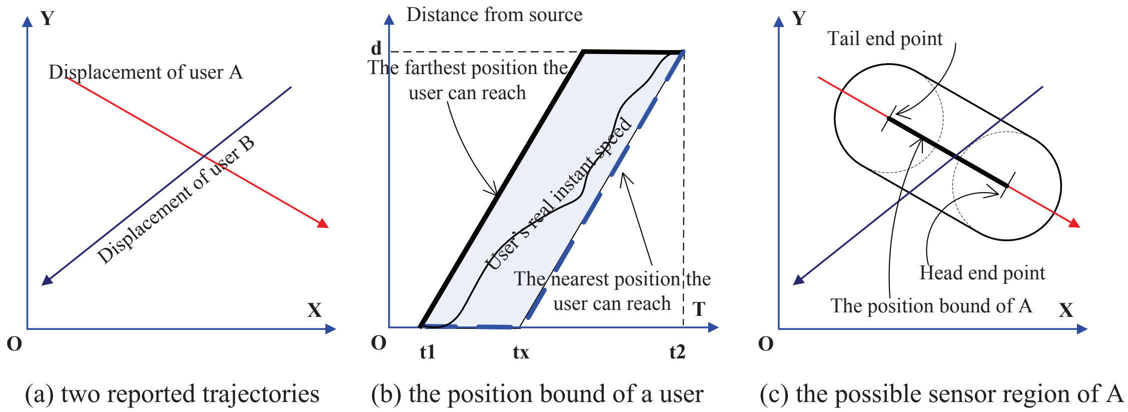


Figure 9.3: Relation between two users' trajectories.

function, which restricts a user's possible position at a given time t by indicating the farthest and nearest locations of where the user could have reached. If the user moves along a line, there are two end points of the position bound. The first point is used to describe that the user waits at the beginning, then moves to the destination with maximum speed; the second point is used to describe that the user first moves with maximum speed, then waits at the destination. As a result, the feasible position of a user at time t will be a segment.

Definition 9.1. For any displacement, going along the moving direction, we name the farthest position that the user can get to at time t as the head end point, and the corresponding sensor area as head sensor area; we name the nearest position as the tail end point and call its sensor area as the tail sensor area.

Fig. 9.3(b) illustrates the feasible position bounds of one user. The user's real moving pattern could be any lines inside the parallelogram, except that the slope of the line must be smaller or equal to the slope of bounds. Since each user is equipped with an encounter sensor, the shape of the possible sensor region of a user at any time is similar to an *ellipse*, which is the union of a group of circles whose center is located at the possible position segment, as shown in Fig. 9.3(c).

TBEP algorithm. As one of the key building blocks of our solution, TBEP

Algorithm 24 Trajectory-based encounter prediction (TBEP) algorithm

Input:$\{(x_i^A, y_i^A, t_i^A)\}$, $\{(x_i^B, y_i^B, t_i^B)\}$, S_{max}, R
Output: EC
Run $PTB(\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}, S_{max}, R)$ to obtain SF_H and SF_T.
if $IT(\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}, R)$ can encounter in corresponding time **then**
    $EC.Type$ =“must meet”, $EC.Time$ = corresponding time
else
    if $IT(\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}, R)$ cannot have encounter in corresponding time **then**
        $EC.Type$ =“cannot meet”, $EC.Time$ = corresponding time
    else
        $EC.Type$ =“may meet”, $EC.Time$ = corresponding time
Sort ER’s tuples according to encounter type and time.
Return: encounter conditions

(i.e. Algorithm 24) presents the way for the server to determine the encounter-type of the given trajectories, which can be further used to determine whether abnormal events occur. After receiving the respective displacement of two users, the server first uses the position function ($\vec{p}()$)) to estimate the latest moving time and the fastest arriving time of a user. Then, the server applies the Possible Trajectory Boundary function($PTB()$) to compute the users’ feasible position bounds in X-Y-T space as: the space function of A’s head end point, $SF_H(A)$; the space function of A’s tail end point, $SF_T(A)$; the space function of B’s head end point, $SF_H(B)$; the space function of B’s tail end point, $SF_T(B)$.

Theorem 9.2. *If all of the distances of* $\{SF_H(A),\ SF_H(B)\}$, $\{SF_T(A),\ SF_H(B)\}$, $\{SF_H(A),\ SF_T(B)\}$ *and* $\{SF_T(A),\ SF_T(B)\}$*) are greater than R in this time period, the two users cannot meet with each other, and therefore the server should not contain the corresponding encounter records, as shown in Fig. 9.4(a)*

Proof. It is obvious since all of the distance between the two users are greater than the R. Note that, in Fig. 9.4, the parallelograms represent the feasible position regions of users. Any line inside the region, whose slope is small or equal to the maximum speed, may be the real instant position of the user. Fig. 9.4(a) two trajectories cannot have encounter records. Fig. 9.4(b) two trajectories may have encounter records.

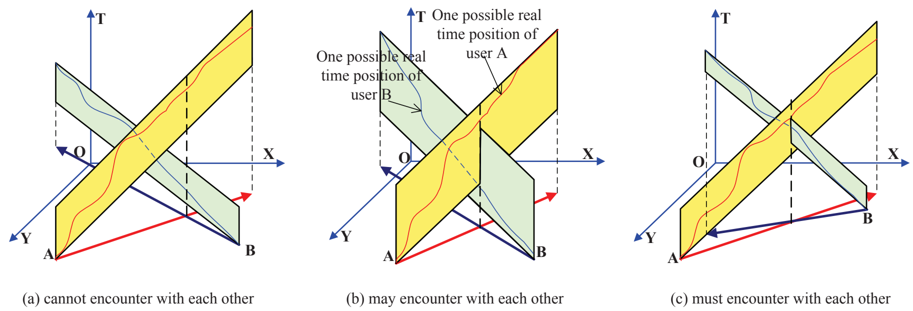


Figure 9.4: Illustration of encounter-type in X-Y-T space.

Fig. 9.4(c) two trajectories must have encounter records. □

Theorem 9.3. *If some of the distances, but not all of the distances of* $\{SF_H(A), SF_H(B)\}$, $\{SF_T(A), SF_H(B)\}$, $\{SF_H(A), SF_T(B)\}$ *and* $\{SF_T(A), SF_T(B)\}$*) are greater than* R *in this time period, the two users may meet with each other.*

Proof. It is also obvious since some trajectories of the two users can encounter with each other and some cannot. □

Theorem 9.4. *If all of the distances of* $\{SF_H(A), SF_H(B)\}$, $\{SF_T(A), SF_H(B)\}$, $\{SF_H(A), SF_T(B)\}$ *and* $\{SF_T(A), SF_T(B)\}$*) are less than* R*, or are equal to* R *in this time period, the two users cannot meet with each other, and therefore the server should contain the corresponding encounter records.*

Proof. The feasible position bounds of the two users are two parallelograms in X-Y-T space. We name the feasible position bound in X-Y-T space as the *Feasible Position Region*. Since space is continuous and all of the edges satisfy the conditions for encountering with each other, any pair of lines inside the parallelograms must encounter with each other. □

The relation between R and the distance of two lines can be computed by the Intersection function($IT()$).

Table 9.2: Table of notation for helper functions

$\vec{p}(t)$	user's position at t in system's coordinates
S_{max}	the maximum speed
(x_i^A, y_i^A, t_i^A)	a piece of displacements of user A in T
$(x_1^A(t), y_1^A(t))$	location where A started the displacement
$(x_2^A(t), y_2^A(t))$	location where A ended the displacement
t_1^A	time that user A began the displacement
t_2^A	time that user A finished the displacement
d	length of a displacement
$(x^A(t), y^A(t))$	possible location of A at given time t
(x^A, y^A)	the location of A at any time
R	sensing range
a,b,c,d	temporary parameters

9.5 Technical Details

In this section, we will present the technical details about our solution. First, we will introduce the position function which we used to indicate the feasible position of a user at a given time instance in X-Y coordinates. The possible trajectory boundary function is the extension of the position function in X-Y-T space, which will be discussed at the second part of this section. After that, the intersection function will be presented. We use the intersection function to estimate the encounter conditions of two users' feasible positions. In the last part of the section, we will define what an abnormal event is and will explain why we classified the entire encounter conditions into three groups: "probably honest", "probably dishonest", and "uncertain."

9.5.1 Position Function

Given one displacement of a user's trajectory, the location of the user can be presented by $\vec{p}(t)$, as shown in Fig. 9.3(b). Suppose that a user reported one displacement from location $\vec{p}(t_1)$ to $\vec{p}(t_2)$ at time from t_1 to t_2 ($t_1 < t_2$), and the length of this displacement equals d. We also assume that the speed range of a user is $[0, \vec{S}_{max}]$.

$$\vec{p}(t) = \begin{cases} [\vec{p}(t_1), \vec{p}(t_1) + \vec{S}_{max}(t - t_1)], & t \in \left[t_1, t_2 - \frac{d}{|\vec{S}_{max}|}\right] \\ [\vec{p}(t_1) + \vec{S}_{max}(t - t_2 + \frac{d}{|\vec{S}_{max}|}), \vec{p}(t_1) + \vec{S}_{max}(t - t_1)], & t \in \left(t_2 - \frac{d}{|\vec{S}_{max}|}, t_1 + \frac{d}{|\vec{S}_{max}|}\right] \\ [\vec{p}(t_1) + \vec{S}_{max}(t - t_2 + \frac{d}{|\vec{S}_{max}|}), \vec{p}(t_2)], & t \in \left(t_1 + \frac{d}{|\vec{S}_{max}|}, t_2\right] \end{cases}$$

In order to simplify the solution, we add a new dimension "time" into the traditional X-Y coordinates. Therefore, Fig. 9.3(a) can be converted to Fig. 9.4. The displacement of each user is represented by a parallelogram in X-Y-T space. The projection of the parallelogram on the X-Y plane is the same segment in Fig. 9.3(a). The lines, which are parallel with the t-axis, represent the situation of not moving. Because of the speed limitation, the slope between any two points in the same parallelogram should be less or equal to the maximum speed. The real trajectory of the user in X-Y-T space will be a curve in the parallelogram. Hence, if A and B meet with each other, their real trajectories must intersect in X-Y-T space. In X-Y-T space, the position function $\vec{p}(t)$ will be converted to the *possible trajectory boundary function (PTB())* .

9.5.2 Possible Trajectory Boundary Function

Possible Trajectory Boundary (PTB()) function provides the two position bounds of a user in X-Y-T space. It has four inputs: $\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}, S_{max}$ and R. The user first moves with maximum speed to finish the reported displacement, then stays still at the end of the displacement; the user can also wait first at the beginning point, then finish the displacement just on time. Therefore, there are two trajectory boundary functions. The head boundary function is:

$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - t_1^A}{d/S_{max}}$$

The tail boundary function is:

$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - (t_2^A - d/S_{max})}{d/S_{max}}$$

Given a pair of displacements, if we know that the users covered the displacements at a constant speed, we can use the *intersection function (IT())* to check whether an encounter happened.

9.5.3 Intersection Function

We use the Intersection IT() function to determine whether the two given users can encounter each other. It has three inputs: $\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}$ and R. Assume that there are two users named A and B. They all report one displacement, and we also assume that they all moved at maximum speed S_{max}. We assume that the beginning time is t_1^A, and the end time is t_2^A. At the same time, the initial position is (x_1^A, y_1^A), and the end of the displacement is (x_2^A, y_2^A). Therefore, the displacement of A can be represented as $((x_1^A, y_1^A, t_1^A), (x_2^A, y_2^A, t_2^A))$. Similarly, we also have B's displacement as $((x_1^B, y_1^B, t_1^B), (x_2^B, y_2^B, t_2^B))$. Now we want to compute when and where these two users can generate the encounter records. Firstly, the trajectory of a user can be represented as follows in a 3-D space, where the x-axis is one possible moving direction and the y-axis is the perpendicular direction to the x-axis. The third direction is time.

$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - t_1^A}{t_2^A - t_1^A}$$

The trajectory of B from t_1^B to t_2^B can be represented as follows:

$$\frac{x^B - x_1^B}{x_2^B - x_1^B} = \frac{y^B - y_1^B}{y_2^B - y_1^B} = \frac{t^B - t_1^B}{t_2^B - t_1^B}$$

Hence, to a given time t, whether a piece of encounter record will be generated or not can be determined by the truth value of following formula:

$$(x^A(t) - x^B(t))^2 + (y^A(t) - y^B(t))^2 \leq R^2$$

After using t to represent $x^A(t)$, $x^B(t)$, $y^A(t)$ and $y^B(t)$, we can get:

$$(a^2 + c^2)t^2 + 2(ab + cd)t + (b^2 + d^2 - R^2) \leq 0,$$

where:

$$a = \frac{x_2^A - x_1^A}{t_2^A - t_1^A} - \frac{x_2^B - x_1^B}{t_2^B - t_1^B}$$

$$b = -(\frac{x_2^A - x_1^A}{t_2^A - t_1^A})t_1^A + (\frac{x_2^B - x_1^B}{t_2^B - t_1^B})t_1^B + x_1^A - x_1^B$$

$$c = \frac{y_2^A - y_1^A}{t_2^A - t_1^A} - \frac{y_2^B - y_1^B}{t_2^B - t_1^B}$$

$$d = -(\frac{y_2^A - y_1^A}{t_2^A - t_1^A})t_1^A + (\frac{y_2^B - y_1^B}{t_2^B - t_1^B})t_1^B + y_1^A - y_1^B$$

9.5.4 Abnormal Events

Here, we define what an abnormal event is. From the view of the server, it can observe two things: trajectories and encounter records. The trajectories of two users have three encounter-types: must meet, potentially meet, and cannot meet. The encounter records about two users at a given time can also be classified into three kinds: no encounter, single encounter, and two encounters. Based on the given trajectories, we can classify the observations made by the server into three groups: an "uncertain" group, a "probably honest" group, and a "probably dishonest" group for all of the 9 kinds of combinations. An abnormal case will fall under the "probably dishonest" group.

The "probably honest" group represents the conditions: (i) at a given time t, the position bounds of two users satisfy the condition of "must meet," and there is a pair of accordant encounter records in the server; or, (ii) the trajectories of two users satisfy the case of "may meet", and a pair of consonant encounter records is stored on the server. If both of the two users are honest, the happening of the conditions (i) and (ii) is obvious. However, if one of the users or both of them are the adversaries, then they can only be classified as "probably honest" only if the adversary can correctly guess the identity of the other party. The probability of this is negligible.

The "uncertain" group also consists of two conditions: (i) the trajectories of two users satisfy the case of "may meet", and there is not any encounter record; or, (ii) the trajectories indicate the condition of "cannot meet", and there is no encounter record. We cannot conclude anything about the identities from these two cases. If both of them are honest users, these two conditions can happen; If there are one or two adversaries, the two conditions can still happen with a high probability. For the condition (i), when there is only one adversary, the adversary can report a fake path, which has a spatial intersection with the path of the honest user, and the real path of the adversary does not encounter the honest user's path. If both of them are the adversaries, they just report two spatial intersected paths, but they do not encounter each other in fact; then, the condition (i) happens. As for the condition (ii), if there is a pair of "impossible meet" paths and the real path of one adversary does not encounter the other user, no matter whether the identity of the other one is honest or not, condition (ii) can happen.

Except the above 4 conditions, the remaining 5 conditions belong to the "probably dishonest" group where at least one of the users could be an adversary.

Theorem 9.5. *An adversary is more likely to be associated with an abnormal case than an honest user if the number of real users is greater than that of the adversaries.*

Proof. Assume that p is the probability that two users meet with each other at a given time. Suppose that the number of real users is a, the number of the fake users is b, and $a >> b$. For a real user, the reason for having abnormal data is that he encountered the adversaries' real paths or fake paths. Hence, given the condition that a real user meets with another user, the conditional probability that the given user meets an abnormal event is $2b/(a-1+2b)$. However, given an adversary, if his real trajectory comes across the trajectories of real users or other adversaries' real trajectories, or if the given adversary's fake trajectory meets with the trajectories of real users or other adversaries' fake trajectories, he may encounter inconsistent conditions. Therefore, given the condition that a fake user meets with another user, the conditional probability that the given user potentially meets an abnormal event is 1. $\square$

9.6 Performance Analysis and Evaluation

In this section, we will first describe our simulation setup. Then, our evaluation metrics will be presented. The simulation results and analysis will be given in the last part of the section.

9.6.1 Experimental Setup

We use Matlab to make the simulation. The users' trajectories are generated by Levy walks mobility model [157] and users' real traces [1]

In the simulation based on synthetic data, we first randomly generate 50 test groups, and each group contains the trajectories of 50 users. We let the period of the location updates be 60 sec, and we observe each user for $6,000$ units of time. We

[1] The real data comes from CRAWDAD, which is a community resource for archiving wireless data at Dartmouth. The data set we used names ncsu/mobilitymodels/GPS2009072392200907232009-07-21. The data was collected by GPS from NCSU, North Carolina, USA, 2009.

set the size of the target area as 500×500 distance units. The silence time period is changed from 0 to 10 seconds. The mobility model's parameter α equals 1, and β also equals 1. We also set the minimum flight length as 60 distance units, and we set the maximum flight length as 100 distance units. When users get the boundary of the target region, we use *reflection* to ensure that the total number of users do not change.

We assume that the encounter sensor range is R. In the simulation, we mainly use $R = 5$ distance units. The maximum speed of users is represented by S_{max}, and we mainly let it be 3 distance units per second. In each time unit, the user randomly picks a speed from 0 to S_{max} to be the real moving speed in the time unit. During simulation, we randomly select N_{Real} users' traces as the real users' trajectories. We select N_{AR} users' traces as the fake trajectories of adversaries, and we also select N_{AF} users' trajectories as the real hidden trajectories of the adversaries $N_{AR} = N_{AF}$. After dividing the trajectory data set, we first compute the encounter records if adversaries report their real traces, which also means there are no adversaries.

During the computation of encounter records, we assume that each user finishes the displacements by the average speed in this segment. When the distance between two users is less or equal to the sensorial range R, we record the identities of these two users, and we also record the beginning and the end time of this encounter record. Then, we observe the displacements of users for a period of time. In each period, we use the reported trajectories of users to check whether there are some abnormal cases. If we find an abnormal case, we will give a penalty to both of the users. After observing the traces for a pre-defined period of time, we rank the suspecting list in descending order and pick the first N_AR users as the adversaries.

In the simulation based on real data, we have the moving trajectories of 35 users. We randomly set the identity of users as real users' trajectories, adversaries' real trajectories, and adversaries' fake trajectories. For example, if there are 5 adversaries, then we randomly choose 5 from the data set as their real moving paths, and we randomly choose another 5 as their reported fake trails. The data set, which we used, records the positions of a user in $2,604$ reporting units of time. The setting for the rest of the parameters is the same as the setting in the synthetic data based simulation.

Considering the length of observation, the adversaries' density, the users' density, and the length of users' sensorial ranges will affect the accuracy of the adversary detection algorithm; we use the simulations to analyze the following relations:

1. The relationship between the length of observation time and the accuracy of our proposed detection algorithm.
2. The relationship between adversaries' density and the accuracy of our proposed detection algorithm.
3. The relationship between users' density and the accuracy of our proposed detection algorithm.
4. The relationship between the length of users' sensorial ranges and the accuracy of our proposed detection algorithm.

9.6.2 Evaluation Metric

Our evaluation was independently made in two different assumptions, and therefore two metrics were used in our evaluation. In the first part of our simulation, we assume that the central server somehow knows the total number of adversaries. Although this assumption seems unrealistic, our simulation results are still valuable if the social scientists can predict the average number of adversaries by some statistic model since the auxiliary information will increase the speed and the accuracy of detecting the adversaries. The first metric we used to evaluate our algorithm is termed *Accuracy*, which is computed as the following:

$$Accuracy = \frac{\text{Number of successfully detected}}{\text{Total number of adversaries}}$$

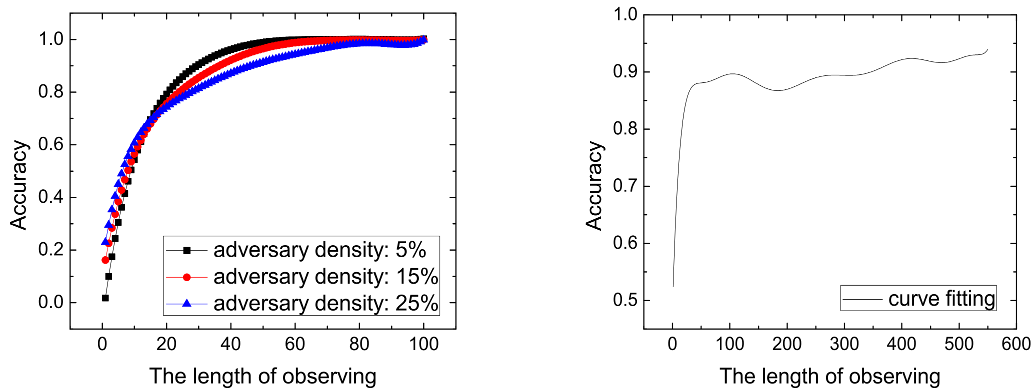
The second part of our evaluation is based on the assumption that there is no auxiliary information about the number of adversaries. During the simulation, we use the clustering algorithm to classify the users into two groups based on their suspected degree, which is computed by our algorithm. The metrics we applied in this part are False Positive Rate (FPR) and False Negative Rate (FNR).

$$FPR = \frac{\text{FP}}{\text{FP+TN}}, \quad FNR = \frac{\text{FN}}{\text{TP+FN}}$$

where FP is the number of false positive, FN is that of false negative, TP represents the total amount of true positive, and TN represents that of true negative.

9.6.3 Simulation Results

Fig. 9.5 indicates the relationship between the accuracy of our proposed algorithm and the length of observing time. In the synthetic data-based simulation, which is shown in Fig. 9.5(a), we randomly select 40 users from each test group as the real ones, and we also randomly pick 5%, 15%, and 25% of users as the adversaries. We use our algorithm to detect the wormhole makers by observing the traces of users for a period of time (100×60 seconds). We repeat our experiments 50 times and use the average value of results to draw Fig. 9.5. From the simulation results, we could find that the accuracy of the proposed algorithm is increased with the growth of observing time. From an analytical point of view, as the observing time increases, there will be more intersections between users' trajectories in temporal space. Hence, we can obtain more "witness points", which may contain the abnormal cases.



(a) synthetic data (b) real data

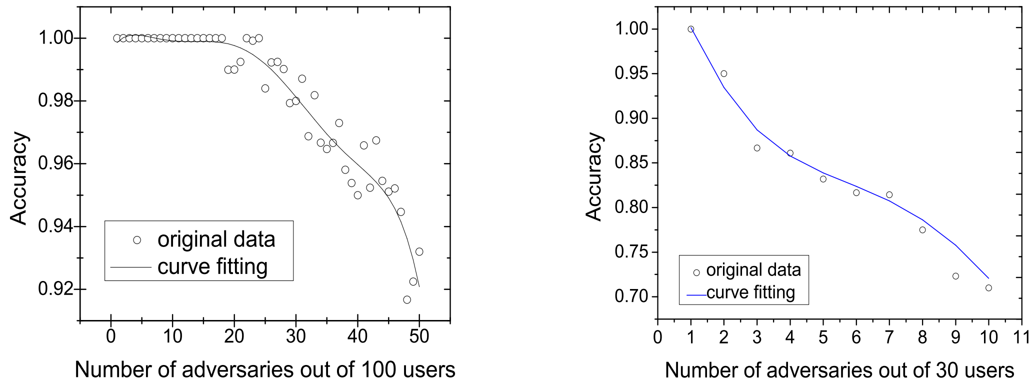
Figure 9.5: Observing time vs. accuracy.

Fig. 9.5(b) shows the relationship between the length of observing and the accuracy in the real trace. In our experiment, we randomly select 25 users as the real ones and 5 users as the adversaries. We set the sensor range as 350 distance units. We repeat our experiment 50 times and get Fig. 9.5(b) as our result. This result is consistent with the synthetic data's result.

However, having more observing time also costs more resources to compute. In order to compare the effectiveness of other components, in the later simulation, we always use 50×60 seconds as the default length of observing time, and we use 350 units of time as the default length of observing time in real trace-based experiments.

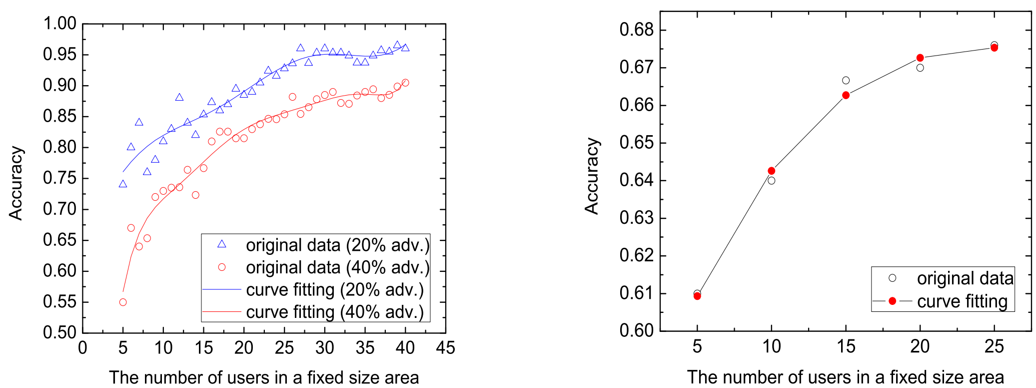
The ratio of adversaries out of the total number of participants can be used to evaluate the influence on accuracy. Fig. 9.6(a) represents the relation between this ratio and the detection accuracy by using synthetic traces. With the increasing number of adversaries, the accuracy of detection goes down since the trust of users becomes less and less. Fig. 9.6(b) is our experiment based on real data. We repeat our simulation 30 times.

The density of users in a region is also an important factor which influences the accuracy of detection. Fig. 9.7(a) illustrates the relation between participant density and the accuracy. In the simulation, we randomly generate 50 test data sets, and



(a) synthetic data (b) real data

Figure 9.6: Adversary density vs. accuracy.

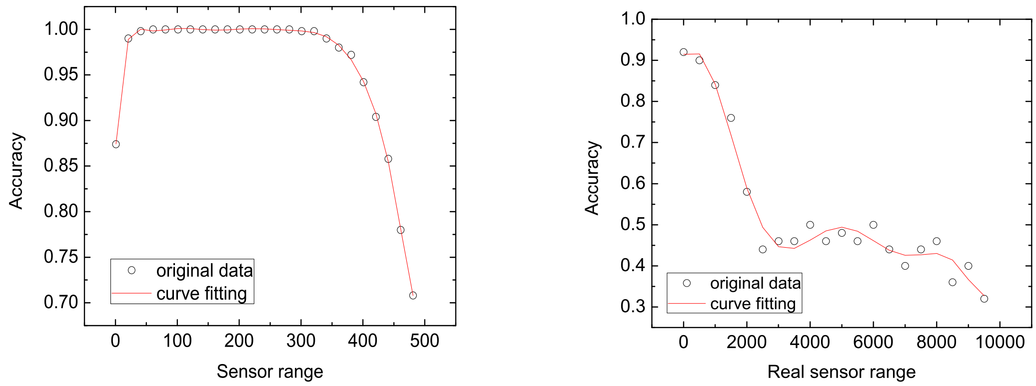


(a) synthetic data (b) real data

Figure 9.7: User density vs. accuracy.

we respectively check the accuracy from density equaling 5 per 500×500 area to 40 per 500×500 area. Also, we assume that the percentage of adversaries is 20% and 40%. Fig. 9.7 is the average result of our simulation. From the image, we can realize that a higher density of users will bring more accuracy, which can be verified by our experiment results based on real traces, which is shown in Fig. 9.7(b).

Figs. 9.8 and 9.9 show the relationship between the encounter sensor range and the detection accuracy. In the synthetic data-based simulation, we select 30 users' trajectories as the traces of real users in each test group, and we select another 10 users' trajectories as the real traces of the adversaries. Then, we obverse 60×60 seconds. Since our proposed wormhole detection and isolation algorithm is based on

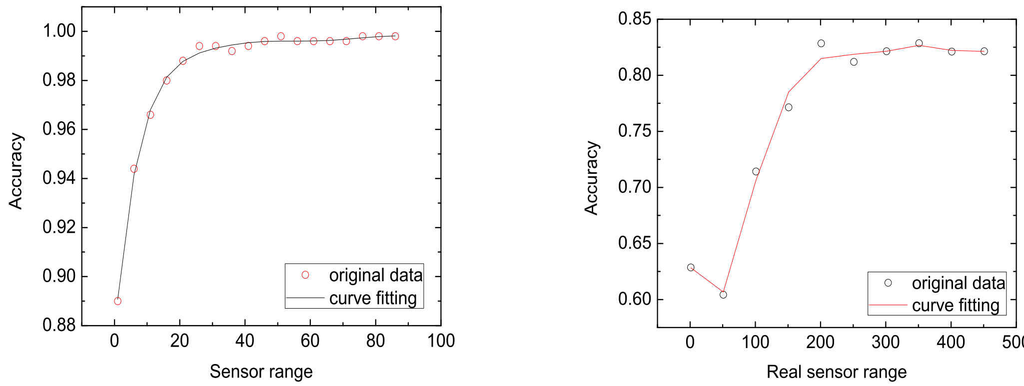


(a) synthetic data (b) real data

Figure 9.8: Sensor range vs. accuracy (overall).

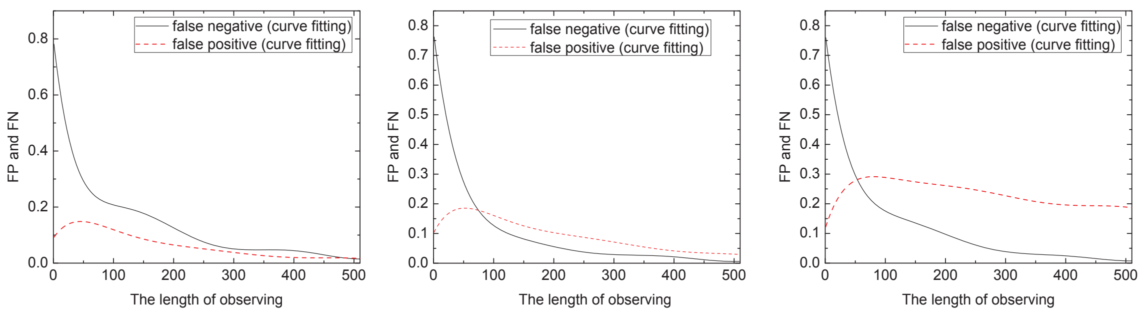
abnormal cases in the encounter area, the accuracy of our algorithm will be increased when there are more encounter conditions. Using a larger sensorial range R will provide more chances for encountering the other users. However, if the value of R exceeds a value, the accuracy will become less since there may always be abnormal events in the region. In the simulation, we first let R change from 1 distance unit to 500 distance units. From Fig. 9.8(a), we can clearly see that the accuracy is increased with the growth of the size of the sensor region in the beginning part. Then, the accuracy maintains stabilization, and it goes down after R is larger than 320. In order to clearly observe the growing pattern in the beginning part, we make another simulation. This time, we let R vary from 1 to 90 distance units, and the simulation result is shown in Fig. 9.9(a). The results of the real traces are respectively shown in Figs. 9.8(b) and 9.9(b). Each data point in the picture is the average value of 50 test results.

In the second part of our simulation, we assume that the total number of adversaries is not available to the center server. After observing the collected data for a while, the server uses the "max cluster" algorithm on the suspected degree data set. In order to increase the difference between honest users and adversaries' suspected degrees, we increase the length of our simulation's time to at least 300 units of time



(a) synthetic data (b) real data

Figure 9.9: Sensor range vs. accuracy (beginning part).



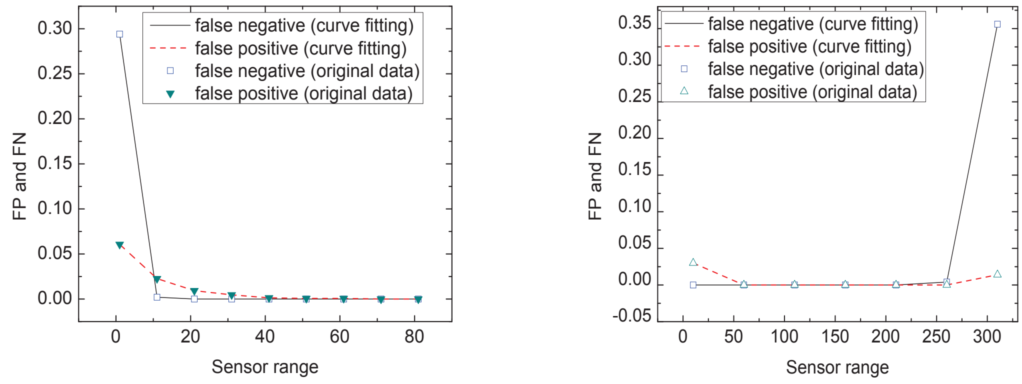
(a) 5% adversary density (b) 15% adversary density (c) 25% adversary density

Figure 9.10: Observing time vs. false positive and false negative.

(300×60 seconds). We repeat each simulation 50 times.

Fig. 9.10 shows the relationship between the length of observation time and false positive, false negative. All of the simulation settings are the same as the configurations in Fig. 9.5, except that the observing time is 500 units of time. From the simulation results, we can clearly see that observing longer will increase the accuracy of detecting the adversaries, and high adversary density requires more observing time to get an accurate result.

Fig. 9.11 indicates the relation between the encounter sensor range and the accuracy in false positive and false negative formations. Similar to previous evaluations, we made two simulations. We first checked the change pattern in the accuracy from 1 to 81 distance units, and then we tested the overall accuracy changing



(a) beginning part (b) overall

Figure 9.11: Sensor range vs. false positive and false negative.

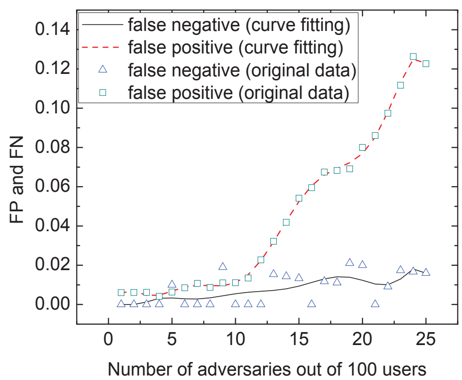


Figure 9.12: Adv. density vs. FP & FN.

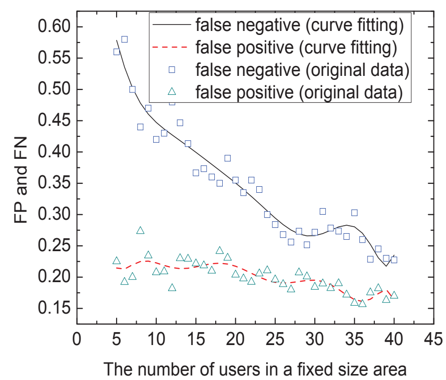


Figure 9.13: User. density vs. FP & FN.

pattern in which the sensor range varied from 10 to 310. The simulation results are consistent with our first part's results.

Fig. 9.12 illustrates the relationship between the accuracy and the adversaries' density. In the simulation, we changed the adversary density from 1% to 25%. Our simulation shows that our algorithm has very high accuracy when the adversary density is less than 15%. Higher adversary density requires the server to observe the data for a longer period.

Our last simulation is about the user density in an area of fixed size. Although the adversary density may be the same, a higher user density can cause more chances for having encounters with others. In the simulation, we set the adversary density as

20%. Our simulation results are in Fig. 9.13, which shows that a higher user density can increase the speed of adversary detection, and it can also increase the accuracy of the detection.

In summary, we considered four key factors which can influence the accuracy of our algorithm. We found that observing the trajectories longer will improve accuracy since this increases the probability of abnormal events. If the total number of users is fixed, then increasing the number of adversaries in a target region reduces the accuracy of our algorithm. When the user density becomes larger, the accuracy of detection will also be increased since it increases the probability of having encounters in a fixed size region. We also determine that increasing the users sensing range will also improve the accuracy of our detection algorithm.

9.7 Further discussion

Here, we discuss some additional issues regarding our solution:

Adversary Reporting Behavior. In this chapter, we consider that the adversaries continuously report some fake locations. An alternative is for the adversary to mix some real locations with fake locations to try to avoid detection. However, the server can easily detect the abnormal data: the distance between two reported locations cannot be larger than the maximum speed. If the adversaries only report some fake locations, which are close to the real ones, the fake paths may not result in a shorter, nonexistent path. Hence, we only consider the case that the adversaries continuously use fake trajectories.

Effect of the Number of Users on Detection. Our method does not require a minimum number of users in a fixed size area in order to detect dishonest users, but we do require that the number of honest users must be larger than that of the dishonest. From our simulation part, we have already seen that the smaller the adversary density is, the better our method's performance is.

Adversary Framing the Honest User. Instead of creating a fake and short path, the adversary can also try to frame the honest users by maliciously reporting a false encounter. Our method can deter this condition since our inconsistency penalty counts on both users. Framing other users will also expose the adversary himself.

Adversary's Long, Fake Path. One attack that is hard to defend against is when the adversary reported a nonexistent, relatively longer path between two locations. Later, as new trajectories are added to the server, that nonexistent trajectory may become part of the shortest path between the two locations; the case rarely happens if the user density is large enough: the probability of encountering with nobody is small. In order to solve the case, during the phase of building the trajectory map, we can only use the road segments which have already been covered by many users. By this way, the unconcluded adversary cannot launch the attack.

9.8 Chapter Summary

For system designer, once the functioning of your application depends on the data contributed by users, it is necessary to incorporate data verification mechanism since attackers may hide among ordinary users. In this chapter, we consider the problem of wormhole attacks in cooperative trajectory mapping. We propose a witness-based detection and isolation algorithm, which identifies and removes fake paths.

Here are the contributions of this chapter:

1. We are the first to explore the problem of fake trail attacks in the context of cooperative trajectory mapping.
2. We propose a witness-based detection algorithm that is effective against multiple, non-collusive, adversaries. Our algorithm acts as a deterrent to

adversaries seeking to attack the system.

3. Extensive simulation experiments were used to validate our solution.

CHAPTER 10

DISSERTATION CONCLUSION

This thesis presents in-depth studies of the social information-assisted application design and its related security and privacy issues. Due to the diversity of social features and application systems, systematically finding security/privacy vulnerability becomes a challenging problem.

In this thesis, we first utilize different social information to design three systems, and then we inspect their security and privacy vulnerabilities from three aspects: data publishing, data accessing, and data usage. The corresponding countermeasures are provided to defend these newly-found attacks.

10.1 Summary of Contributions

Incorporating social information into system design has already become a new trend. However, social information is a general concept, and there are many types of social features, such as mobility pattern, online chatting frequency, and social closeness. Depending on application scenario and service purpose, many social information-assisted applications have been designed by using different social features. Due to the diversity of these applications, each aspect related to social information-assisted application design contains security/privacy vulnerabilities for certain applications. We design new attacks and the corresponding defense schemes for each of the three aspects.

This dissertation starts with improving the performances of P2P system, mobile clouds, and cooperative trajectory mapping via using different social features. Through these three systems, we let readers understand the importance and effectiveness of using social information and the complexity and diversity of these applications. Next, we deeply study the security and privacy issues in social information-assisted application design. More specially, we essentially ask three questions:

1. How does a system publish/obtain social information?
2. Who can access the social information in an application?
3. How does these social information-related data been used?

For a given application, by answering these questions, one may find its security or privacy vulnerabilities.

For the first question, which considers the vulnerability from a data publishing aspect, we study the privacy-preserved data publishing in centralized and distributed systems. We identify a new attack by including a time domain into the data anonymization problem, and we propose a new requirement for the privacy protection in distributed scenario, *multi-level privacy*. Several data anonymization algorithms are designed, and extensive simulations show that more data utility can be preserved by applying our algorithms.

For the second question, which considers the vulnerability from a data accessing aspect, we study how attackers can benefit from knowing a target's social information. We create a powerful location privacy disclosure attack by exploring users' co-appearance frequency with their friends. We theoretically analyze the problem and provide a fake location-based countermeasure.

For the last question about data usage, we use Sybil attacks in mobile social networks and Wormhole attacks in cooperative trajectory mapping to show that trust mechanism or data verification must be included in a social information-assisted design. Once original social information's accuracy is problematic, the corresponding application's quality of service cannot be guaranteed. To defend these two attacks, for each system, we design different verification approaches.

10.2 Future Research

Since the social information-assisted techniques are increasingly popular and the performance of any existing system involved with human beings potentially can be improved by incorporating social information, how to securely and privacy-preserved apply the sociality is becoming more important and inevitable. In the future research, we plan to continue working on improving the security and privacy protections of social information-assisted systems.

One research direction is to build a set of general prototypes for all social information-assisted systems. As we have pointed out in this dissertation, one reason for having some much vulnerability in these applications is the diversity. Identifying security/privacy vulnerabilities will become much easier if we could abstract a system into a set of modules with different functioning. By using the prototypes, we can inspect the risks in an individual module or their combinations. Once a new attack is found, the defense mechanism of the corresponding module can be adjusted, and all applications shared this type of modules can immediately determine their security risks and accordingly update their systems.

Another future research direction is to implement the proposed systems on real devices. So far, we mainly use synthetic or real data-based simulation to evaluate the performance of our approaches. In future, we would like to test them on real systems. However, since our problems mainly involve distributed system, mobile device, cloud, networking, security, and privacy, the real system-based experiments need to integrate all these techniques, which is quite challenging.

Recently, the concept of smart cities is becoming more and more popular, and we plan to introduce sociality-based approaches into the scenarios of smart cities. Unlike the systems that we have worked on, not all devices in smart cities are controlled or carried by human beings; instead, the main objective of these devices is to serve people. Based on the activity patterns, new scheduling algorithms can be designed for better controlling certain devices, such as traffic lights or public transportation schedule, in a smart city. In addition, there will be more security and privacy issues. For example, we will study the problem, like how to generate location proofs for a vehicular trajectory without the loss of other users' location privacy. The tradeoff among utility, security, and privacy has not been well studied yet, especially when social information is involved. In the future, we will continue work on this problem.

PUBLICATIONS

Journal Publications

1. **Wei Chang** and Jie Wu, "On the privacy preserved data publishing of evolving online social networks," Journal of Information Privacy and Security (IPS), Vol. 12, No. 1, 2016, 14-31.

2. **Wei Chang** and Jie Wu, "Progressive or Conservative: Rationally Allocate Cooperative Work in Mobile Social Networks," IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 26, No. 7, 2015, 2020-2035.

3. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Improving Cooperative Trajectory Mapping Applications with Encounter-based Error Correction," International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), Vol. 29, No. 1, 2014, 68-89.

4. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Friendship-based Location Privacy in Mobile Social Networks," International Journal of Security and Networks (IJSN), Vol. 6, No. 4, 2012, 226-236.

5. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Wormhole Defense for Cooperative Trajectory Mapping," International Journal of Parallel, Emergent and Distributed Systems, Vol. 27, No. 5, 2012, 459-480.

Conference Publications

6. Huangyang Zheng, **Wei Chang**, and Jie Wu, "Coverage and Distinguishability Requirements for Traffic Flow Monitoring Systems," Proc. of the IEEE/ACM International Symposium on Quality of Service (IWQoS), June, 2016. (**Best paper award**)

7. **Wei Chang**, and Jie Wu, "SocialVoD: a Social Feature-based P2P System," Proc. of the 44th International Conference on Parallel Processing (ICPP), September, 2015.

8. **Wei Chang**, and Jie Wu, "Reliability Enhanced Social Crowdsourcing," Proc. of the IEEE Global Communications Conference (GlobeCom), December, 2015.

9. Qin Liu, Guojun Wang, Jie Wu, and **Wei Chang**, "User-Controlled Security Mechanism in Data-Centric Clouds," Proc. of the 17th IEEE International Conference on High Performance Computing and Communications (HPCC Invited paper), August, 2015.

10. **Wei Chang** and Jie Wu, "Strengthening Data Privacy During Propagation," Proc. of the ACM MobiCom Workshop on Security and Privacy Aspects of Mobile Environments (ACM MobiCom-SPME), September 2014.

11. **Wei Chang**, Jie Wu, Chiu C. Tan, and Feng Li, "Sybil Defenses in Mobile Social networks," Proc. of the IEEE Global Telecommunications Conference (GLOBECOM), December, 2013.

12. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Cooperative Trajectory-based Map Construction" Proc. of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), June, 2012.

13. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Encounter-based Noise Cancelation For Cooperative Trajectory Mapping," Proc. of the 10th IEEE Pervasive Computing and Communication Conference (PerCom), March, 2012.

14. **Wei Chang**, Jie Wu, and Chiu C. Tan, "Enhancing Mobile Social Network Privacy," Proc. of the IEEE Global Telecommunications Conference (GLOBECOM), December, 2011.

Book Chapters

15. **Wei Chang** and Jie Wu, "A New View of Privacy in Social Networks: Strengthening Privacy During Propagation," *Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security*, Ed. Brij Gupta, Dharma P. Agrawal, and Shingo Yamaguchi, IGI Global, 2016, 28-51.

16. **Wei Chang** and Jie Wu, "A Survey of Sybil Attacks in Networks," *Sensor Networks for Sustainable Development*, Ed. Mohammad Ilyas, CRC Press, 2014.

Submitted

17. **Wei Chang**, Huanyang Zheng, Jie Wu, Chiu C. Tan, and Haibin Ling, "Environmental Evidence-Assisted Vehicular Data in Smart Cities," is submitted to *Smart Cities: Foundations, Principles and Applications*, Ed. Houbing Song, Wiley.

18. **Wei Chang**, Wei-Shih Yang, and Jie Wu, "On the Impacts of Correlated Friends in Social-Crowdsourcing," is submitted to Journal of Parallel and Distributed Computing.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] H. Shen, Y. Lin, and H. Chandler, "An Interest-Based Per-Community P2P Hierarchical Structure for Short Video Sharing in the YouTube Social Network," in *IEEE ICDCS*, 2014, pp. 298–307.

[2] "YouTube Statistics," https://www.youtube.com/yt/press/statistics.html, accessed: 2014-10-20.

[3] "Vimeo Wiki," http://en.wikipedia.org/wiki/Vimeo, accessed: 2014-11-16.

[4] "5 Reasons to Choose Vimeo Instead of YouTube," https://www.youtube.com/yt/press/statistics.html, accessed: 2014-11-16.

[5] "Top 10 Most Popular YouTube Channels," http://www.ignitesocialmedia.com/youtube-marketing/top-10-most-popular-youtube-channels/, accessed: 2014-09-16.

[6] J. Lu and J. Callan, "Content-based peer-to-peer network overlay for full-text federated search," in *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)*. C.I.D., 2007, pp. 490–509.

[7] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" *ACM SIGCOMM*, vol. 37, no. 4, pp. 133–144, 2007.

[8] X. Cheng and J. Liu, "Nettube: Exploring social networks for peer-to-peer short video sharing," in *IEEE INFOCOM*, 2009, pp. 1152–1160.

[9] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "HyperCuPhypercubes, ontologies, and efficient search on peer-to-peer networks," in *Springer AP2PC*, 2003, pp. 112–124.

[10] S. Annapureddy, C. Gkantsidis, P. Rodriguez, and L. Massoulie, "Providing video-on-demand using peer-to-peer networks," in *IPTV workshop*, 2006.

[11] B. Cheng, L. Stein, H. Jin, X. Liao, and Z. Zhang, "GridCast: Improving peer sharing for P2P VoD," *ACM TOMCCAP*, vol. 4, no. 4, p. 26, 2008.

[12] B. Li, M. Ma, Z. Jin, and D. Zhao, "Investigation of a large-scale P2P VoD overlay network by measurements," *Springer PPNA*, vol. 5, no. 4, pp. 398–411, 2012.

[13] E. Rosas, N. Hidalgo, M. Marin, and V. Gil-Costa, "Web search results caching service for structured P2P networks," *FGCS*, vol. 30, pp. 254–264, 2014.

[14] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *ACM SIGCOMM IMC*, 2007, pp. 1–14.

[15] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser, "Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks," in *ACM WWW*, 2003, pp. 536–543.

[16] A. Löser, W. Siberski, M. Wolpers, and W. Nejdl, "Information integration in schema-based peer-to-peer networks," in *Springer AISE*, 2003, pp. 258–272.

[17] K. Lewis, M. Gonzalez, and J. Kaufman, "Social selection and peer influence in an online social network," *PNAS*, vol. 109, no. 1, pp. 68–72, 2012.

[18] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *IEEE IWQoS*, 2008, pp. 229–238.

[19] X. Cheng, J. Liu, and C. Dale, "Understanding the characteristics of internet short video sharing: A YouTube-based measurement study," *IEEE TMM*, vol. 15, no. 5, pp. 1184–1194, 2013.

[20] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster, "The small world of

file sharing," *IEEE TPDS*, vol. 22, no. 7, pp. 1120–1134, 2011.

[21] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[22] H. Zheng, Y. Wang, and J. Wu, "Optimizing Multi-copy Two-hop Routing in Mobile Social Networks," in *IEEE SECON 2014*, 2014, pp. 573–581.

[23] "BOINC: Open-source software for volunteer computing," https://boinc.berkeley.edu/, accessed: 2014-11-16.

[24] "Folding Project," https://folding.stanford.edu/, accessed: 2014-11-16.

[25] "Amazon mechanical turk," https://www.mturk.com/mturk/, accessed: 2014-11-16.

[26] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay, "Cascade: Crowdsourcing taxonomy creation," in *ACM SIGCHI*, 2013, pp. 1999–2008.

[27] D. C. Brabham, *Crowdsourcing*. MIT Press, 2013.

[28] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[29] J. J. Horton and L. B. Chilton, "The labor economics of paid crowdsourcing," in *ACM EC*, 2010, pp. 209–218.

[30] D. C. Brabham, "Crowdsourcing as a model for problem solving an introduction and cases," *Convergence: the international journal of research into new media technologies*, vol. 14, no. 1, pp. 75–90, 2008.

[31] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with Mechanical Turk," in *ACM SIGCHI*, 2008, pp. 453–456.

[32] N. Eagle, "txteagle: Mobile crowdsourcing," in *Internationalization, Design and Global Development*. Springer, 2009, pp. 447–456.

[33] "Mturk Pricing," https://requester.mturk.com/pricing, accessed: 2016-7-2.

[34] S. Zhang, J. Wu, and S. Lu, "Minimum Makespan Workload Dissemination in DTNs: Making Full Utilization of Computational Surplus Around," in *ACM MobiHoc*, 2013, pp. 293–296.

[35] C. Koulamas, S. Gupta, and G. J. Kyparisis, "A unified analysis for the single-machine scheduling problem with controllable and non-controllable variable job processing times," *EJOR*, vol. 205, no. 2, pp. 479–482, 2010.

[36] S. A. Slotnick, "Order acceptance and scheduling: a taxonomy and review," *EJOR*, vol. 212, no. 1, pp. 1–11, 2011.

[37] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *JPDC*, vol. 47, no. 1, pp. 8–22, 1997.

[38] M. Thiele, U. Fischer, and W. Lehner, "Partition-based workload scheduling in living data warehouse environments," *Information Systems*, vol. 34, no. 4, pp. 382–399, 2009.

[39] A. L. Rosenberg, "Sharing partitionable workloads in heterogeneous NOWs: greedier is not better," in *IEEE Cluster*, 2001, p. 124.

[40] L. Tassiulas, "Adaptive back-pressure congestion control based on local information," *IEEE TAC*, vol. 40, no. 2, pp. 236–250, 1995.

[41] S. M. Ross, *Introduction to probability models.* Academic press, 2006.

[42] https://www-personal.umich.edu/simmejn/netdata/, accessed: 2014-11-16.

[43] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins, "The web as a graph: Measurements, models, and methods," in *Computing and combinatorics.* Springer, 1999, pp. 1–17.

[44] H. Deng, M. R. Lyu, and I. King, "A generalized Co-HITS algorithm and its application to bipartite graphs," in *ACM SIGKDD*, 2009, pp. 239–248.

[45] A. McCallum, D. Freitag, and F. C. Pereira, "Maximum Entropy Markov Models for Information Extraction and Segmentation," in *ICML*, vol. 17, 2000, pp. 591–598.

[46] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *ACM/IEEE IPSN*, 2010, pp. 279–290.

[47] M. J. Neely and R. Urgaonkar, "Optimal backpressure routing for wireless networks with multi-receiver diversity," *Ad Hoc Networks*, vol. 7, no. 5, pp. 862–881, 2009.

[48] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury, "Did you see Bob?: human localization using mobile phones," in *ACM MobiCom*, 2010, pp. 149–160.

[49] I. Constandache, R. R. Choudhury, and I. Rhee, "Towards mobile phone localization without war-driving," in *IEEE INFOCOM*, 2010, pp. 1–9.

[50] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *ACM SenSys*, 2008, pp. 323–336.

[51] A. Repenning and A. Ioannidou, "Mobility agents: guiding and tracking public transportation users," in *ACM AVI*, 2006, pp. 127–134.

[52] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. A. Landay, "UbiGreen: investigating a mobile tool for tracking and supporting green transportation habits," in *ACM CHI*, 2009, pp. 1043–1052.

[53] S. Carmien, M. Dawe, G. Fischer, A. Gorman, A. Kintsch, and J. F. Sullivan Jr,

"Socio-technical environments supporting people with cognitive disabilities using public transportation," *ACM TOCHI*, vol. 12, no. 2, pp. 233–262, 2005.

[54] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM*, vol. 2, 2000, pp. 775–784.

[55] Y.-C. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm, "Accuracy characterization for metropolitan-scale Wi-Fi localization," in *ACM MobiSys*, 2005, pp. 233–245.

[56] N. Kayastha, D. Niyato, P. Wang, and E. Hossain, "Applications, Architectures, and Protocol Design Issues for Mobile Social Networks: A Survey," *Proceedings of the IEEE*, vol. 99, no. 12, pp. 2130 –2158, 2011.

[57] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "VTrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *ACM SenSys*, 2009, pp. 85–98.

[58] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *ACM SenSys*, 2010, pp. 85–98.

[59] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, L. Girod *et al.*, "Accurate, low-energy trajectory mapping for mobile devices," *ACM NSDI*, 2011.

[60] P. S. Maybeck and G. M. Siouris, "Stochastic models, estimation, and control, Volume I," *IEEE TSMC*, vol. 10, no. 5, pp. 282–282, 1980.

[61] F. Jay and B. Matthew, *The global positioning system and inertial navigation.* McGraw-Hill Professional, 1999.

[62] D. Titterton and J. L. Weston, *Strapdown inertial navigation technology.* Peter Peregrinus Ltd, 2004, vol. 17.

[63] P.-Y. Gilliéron, D. Büchel, I. Spassov, and B. Merminod, "Indoor navigation

performance analysis," in *ENC GNSS*, 2004, p. 30.

[64] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," 1995.

[65] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[66] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE TSP*, vol. 50, no. 2, pp. 425–437, 2002.

[67] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[68] D. Niculescu and B. Nath, "VOR base stations for indoor 802.11 positioning," in *MobiCom*, 2004, pp. 58–69.

[69] M. Youssef, A. Youssef, C. Rieger, U. Shankar, and A. Agrawala, "Pinpoint: An asynchronous time-based location determination system," in *ACM MobiSys*, 2006, pp. 165–176.

[70] N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller, "Anchor-free distributed localization in sensor networks," in *ACM SenSys*, 2003, pp. 340–341.

[71] F. Fitzek, A. Kopsel, A. Wolisz, M. Krishnam, and M. Reisslein, "Providing application-level QoS in 3G/4G wireless systems: a comprehensive framework based on multirate CDMA," *IEEE TWC*, vol. 9, no. 2, pp. 42–47, 2002.

[72] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics reports*, vol. 424, no. 4, pp. 175–308, 2006.

[73] B. Skyrms and R. Pemantle, "A dynamic model of social network formation,"

in *Adaptive Networks*. Springer, 2009, pp. 231–251.

[74] C. Liu and Z.-K. Zhang, "Information spreading on dynamic social networks," *CNSNS*, vol. 19, no. 4, pp. 896–904, 2014.

[75] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *ACM SIGSPATIAL*, 2012, pp. 199–208.

[76] W. Wei, F. Xu, C. C. Tan, and Q. Li, "Sybildefender: Defend against sybil attacks in large social networks," in *IEEE INFOCOM*, 2012, pp. 1951–1959.

[77] L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou $R3579x$?: anonymized social networks, hidden patterns, and structural steganography," in *ACM WWW*, 2007, pp. 181–190.

[78] L. Rossi, M. Musolesi, and A. Torsello, "On the k-Anonymization of Time-varying and Multi-layer Social Graphs," *ICWSM*, 2015.

[79] K. Liu and E. Terzi, "Towards identity anonymization on graphs," in *ACM SIGMOD*, 2008, pp. 93–106.

[80] J. Cheng, A. W.-c. Fu, and J. Liu, "K-isomorphism: privacy preserving network publication against structural attacks," in *ACM SIGMOD*, 2010, pp. 459–470.

[81] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *IEEE ICDE*, 2008, pp. 506–515.

[82] X. Ding, L. Zhang, Z. Wan, and M. Gu, "De-anonymizing dynamic social networks," in *IEEE GLOBECOM*, 2011, pp. 1–6.

[83] Y. Li and H. Shen, "On identity disclosure control for hypergraph-based data publishing," *IEEE TIFS*, vol. 8, no. 8, pp. 1384–1396, 2013.

[84] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," *CSDFPS*, 2007.

[85] C.-H. Tai, P. S. Yu, D.-N. Yang, and M.-S. Chen, “Privacy-preserving social network publication against friendship attacks,” in *ACM SIGKDD*, 2011, pp. 1262–1270.

[86] L. Zou, L. Chen, and M. T. Özsu, “K-automorphism: A general framework for privacy preserving network publication,” *VLDB Endowment*, vol. 2, no. 1, pp. 946–957, 2009.

[87] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava, “Class-based graph anonymization for social network data,” *VLDB Endowment*, vol. 2, no. 1, pp. 766–777, 2009.

[88] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa, “Injecting uncertainty in graphs for identity obfuscation,” *VLDB Endowment*, vol. 5, no. 11, pp. 1376–1387, 2012.

[89] G. Wang, Q. Liu, F. Li, S. Yang, and J. Wu, “Outsourcing privacy-preserving social networks to a cloud,” in *IEEE INFOCOM*, 2013, pp. 2886–2894.

[90] M. E. Dyer and A. M. Frieze, “Planar 3DM is NP-complete,” *Journal of Algorithms*, vol. 7, no. 2, pp. 174–184, 1986.

[91] A. Zhang, X. Xie, K. C.-C. Chang, C. A. Gunter, J. Han, and X. Wang, “Privacy Risk in Anonymized Heterogeneous Information Networks,” in *EDBT*, 2014, pp. 595–606.

[92] J. Wu and Y. Wang, “Social feature-based multi-path routing in delay tolerant networks,” in *IEEE INFOCOM*, 2012, pp. 1368–1376.

[93] W. Feng and J. Wang, “Incorporating heterogeneous information for personalized tag recommendation in social tagging systems,” in *ACM SIGKDD*, 2012, pp. 1276–1284.

[94] L. M. Aiello, A. Barrat, R. Schifanella, C. Cattuto, B. Markines, and

F. Menczer, "Friendship prediction and homophily in social media," *ACM TWEB*, vol. 6, no. 2, p. 9, 2012.

[95] A. Meyerson and R. Williams, "On the complexity of optimal k-anonymity," in *ACM SIGMOD-SIGACT-SIGART*, 2004, pp. 223–228.

[96] L. Sweeney, "k-anonymity: A model for protecting privacy," *IJUFKS*, vol. 10, no. 05, pp. 557–570, 2002.

[97] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM TKDD*, vol. 1, no. 1, p. 3, 2007.

[98] N. Li, T. Li, and S. Venkatasubramanian, "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *ICDE*, 2007, pp. 106–115.

[99] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee, "Toward privacy in public databases," in *Theory of Cryptography.* Springer, 2005, pp. 363–385.

[100] C. Dwork, "A firm foundation for private data analysis," *ACM CACM*, vol. 54, no. 1, pp. 86–95, 2011.

[101] A. Roth and T. Roughgarden, "Interactive privacy via the median mechanism," in *ACM STOC*, 2010, pp. 765–774.

[102] A. Blum, K. Ligett, and A. Roth, "A learning theory approach to noninteractive database privacy," *ACM JACM*, vol. 60, no. 2, p. 12, 2013.

[103] C. Dwork, "Differential privacy," in *Automata, languages and programming.* Springer, 2006.

[104] K. Nissim, R. Smorodinsky, and M. Tennenholtz, "Approximately optimal mechanism design via differential privacy," in *ACM ITCS*, 2012, pp. 203–213.

[105] P. Samarati, "Protecting respondents identities in microdata release," *IEEE TKDE*, vol. 13, no. 6, pp. 1010–1027, 2001.

[106] S. Kisilevich, L. Rokach, Y. Elovici, and B. Shapira, "Efficient multidimensional suppression for k-anonymity," *IEEE TKDE*, vol. 22, no. 3, pp. 334–347, 2010.

[107] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *IJUFKS*, vol. 10, no. 05, pp. 571–588, 2002.

[108] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu, "Utility-based anonymization using local recoding," in *ACM SIGKDD*, 2006, pp. 785–790.

[109] X. Zhang, C. Liu, S. Nepal, C. Yang, W. Dou, and J. Chen, "Combining top-down and bottom-up: scalable sub-tree anonymization over big data using MapReduce on cloud," in *IEEE TrustCom*, 2013, pp. 501–508.

[110] K. Wang, B. C. Fung, and S. Y. Philip, "Handicapping attacker's confidence: an alternative to k-anonymization," *KAIS*, vol. 11, no. 3, pp. 345–368, 2007.

[111] Y. He and J. F. Naughton, "Anonymization of set-valued data via top-down, local generalization," *VLDB Endowment*, vol. 2, no. 1, pp. 934–945, 2009.

[112] X. Xiao, Y. Tao, and M. Chen, "Optimal random perturbation at multiple privacy levels," *VLDB Endowment*, vol. 2, no. 1, pp. 814–825, 2009.

[113] Y. Li, M. Chen, Q. Li, and W. Zhang, "Enabling multilevel trust in privacy preserving data mining," *IEEE TKDE*, vol. 24, no. 9, pp. 1598–1612, 2012.

[114] J. Brickell and V. Shmatikov, "The cost of privacy: destruction of data-mining utility in anonymized data publishing," in *ACM SIGKDD*, 2008, pp. 70–78.

[115] T. Li and N. Li, "On the tradeoff between privacy and utility in data publishing," in *ACM SIGKDD*, 2009, pp. 517–526.

[116] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE TIT*, vol. 37, no. 1, pp. 145–151, 1991.

[117] X. Xiao, K. Yi, and Y. Tao, "The hardness and approximation algorithms for l-diversity," in *ACM EDBT*, 2010, pp. 135–146.

[118] K. LeFevre, D. DeWitt, and R. Ramakrishnan, “Mondrian multidimensional k-anonymity,” in *IEEE ICDE*, 2006, pp. 25–25.

[119] M. Gruteser and D. Grunwald, “Enhancing location privacy in wireless LAN through disposable interface identifiers:A quantitative analysis,” *Mobile Networks and Applications*, vol. 10, no. 3, pp. 315–325, 2005.

[120] L. Huang, K. Matsuura, H. Yamane, and K. Sezaki, “Enhancing wireless location privacy using silent period,” in *IEEE WCNC*, vol. 2, 2005, pp. 1187–1192.

[121] J. Freudiger, M. H. Manshaei, J.-P. Hubaux, and D. C. Parkes, “On non-cooperative location privacy: a game-theoretic analysis,” in *ACM CCS*, 2009, pp. 324–337.

[122] T. Jiang, H. J. Wang, and Y.-C. Hu, “Preserving location privacy in wireless LANS,” in *Mobisys*, 2007, pp. 246–257.

[123] L. Kulik, “Privacy for real-time location-based services,” *SIGSPATIAL Special*, vol. 1, 2009.

[124] C.-Y. Chow and M. F. Mokbel, “Privacy in location-based services: a system architecture perspective,” *SIGSPATIAL Special*, vol. 1, 2009.

[125] Y. Ouyang, Y. Xu, Z. Le, G. Chen, and F. Makedon, “Providing location privacy in assisted living environments,” in *PETRA*, 2008, p. 39.

[126] M. L. Damiani, E. Bertino, and C. Silvestri, “Protecting location privacy against spatial inferences: the PROBE approach,” in *ACM SPRINGL*, 2009, pp. 32–41.

[127] H. Kido, Y. Yanagisawa, and T. Satoh, “An Anonymous Communication Technique using Dummies for Location-based Services,” in *ICPS*, 2005, pp. 88–97.

[128] J. Krumm, “Inference attacks on location tracks,” in *PERVASIVE*. Springer,

2007, pp. 127–143.

[129] B. Hoh and M. Gruteser, “Protecting Location Privacy Through Path Confusion,” in *Securecomm*, 2005, pp. 194–205.

[130] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, “Preserving privacy in GPS traces via uncertainty-aware path cloaking,” in *ACM CCS*, 2007, pp. 161–171.

[131] ——, “Enhancing Security and Privacy in Traffic-Monitoring Systems,” *IEEE Pervasive Computing*, vol. 5, no. 4, pp. 38–46, 2006.

[132] A. Mohaisen, A. Yun, and Y. Kim, “Measuring the mixing time of social graphs,” in *ACM SIGCOMM IMC*, 2010, pp. 383–389.

[133] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove, “An analysis of social network-based sybil defenses,” in *ACM SIGCOMM*, vol. 40, no. 4, 2010, pp. 363–374.

[134] J. Newsome, E. Shi, D. Song, and A. Perrig, “The sybil attack in sensor networks: analysis & defenses,” in *ACM IPSN*, 2004, pp. 259–268.

[135] M. Demirbas and Y. Song, “An RSSI-based scheme for sybil attack detection in wireless sensor networks,” in *IEEE WOWMOM*, 2006, pp. 564–570.

[136] C. Piro, C. Shields, and B. N. Levine, “Detecting the sybil attack in mobile ad hoc networks,” in *IEEE Securecomm*, 2006, pp. 1–11.

[137] D. Quercia and S. Hailes, “Sybil attacks against mobile users: friends and foes to the rescue,” in *IEEE INFOCOM*, 2010, pp. 1–5.

[138] G. Danezis and P. Mittal, “Sybilinfer: detecting sybil nodes using social networks,” in *NDSS*, 2009.

[139] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, “Sybilguard: defending against sybil attacks via social networks,” in *ACM SIGCOMM*, vol. 36, no. 4, 2006, pp. 267–278.

[140] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *ACM CHI*, 2010, pp. 1361–1370.

[141] N. Chiluka, N. Andrade, J. Pouwelse, and H. Sips, "Leveraging trust and distrust for sybil-tolerant voting in online social media," in *ACM PSOSM*, 2012.

[142] P. V. Marsden, "Egocentric and sociocentric measures of network centrality," *Social networks*, vol. 24, no. 4, pp. 407–422, 2002.

[143] J. Wu, *Handbook on theoretical and algorithmic aspect of sensor, ad hoc wireless, and peer-to-peer networks.* Auerbach Publications, 2006.

[144] L. Buttyan and J.-P. Hubaux, *Security and cooperation in wireless networks: thwarting malicious and selfish behavior in the age of ubiquitous computing.* Cambridge University Press, 2007.

[145] L. Buttyán, L. Dóra, and I. Vajda, "Statistical wormhole detection in sensor networks," *Security and Privacy in Ad-hoc and Sensor Networks*, 2005.

[146] N. Song, L. Qian, and X. Li, "Wormhole attacks detection in wireless ad hoc networks: A statistical analysis approach," in *IEEE IPDPS*, 2005.

[147] L. Qian, N. Song, and X. Li, "Detection of wormhole attacks in multi-path routed wireless ad hoc networks: a statistical analysis approach," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 308–330, 2007.

[148] W. Wang and B. Bhargava, "Visualization of wormholes in sensor networks," in *ACM Workshop on Wireless security*, 2004, pp. 51–60.

[149] W. Wang and A. Lu, "Interactive wormhole detection and evaluation," *Information Visualization*, vol. 6, no. 1, pp. 3–17, 2007.

[150] I. Khalil, S. Bagchi, and N. B. Shroff, "LITEWORP: a lightweight countermeasure for the wormhole attack in multihop wireless networks," in *IEEE DSN*, 2005, pp. 612–621.

[151] ——, "MobiWorp: Mitigation of the wormhole attack in mobile multihop wireless networks," *Ad Hoc Networks*, vol. 6, no. 3, pp. 344–362, 2008.

[152] Y. Wang, Z. Zhang, and J. Wu, "A Distributed Approach for Hidden Wormhole Detection with Neighborhood Information," in *IEEE NAS*, 2010, pp. 63–72.

[153] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," *Ad hoc networks*, vol. 1, no. 2, pp. 293–315, 2003.

[154] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *IEEE INFOCOM*, vol. 3, 2003, pp. 1976–1986.

[155] ——, "Wormhole detection in wireless ad hoc networks," *Department of Computer Science, Rice University, Tech. Rep. TR01-384*, 2002.

[156] S. Čapkun, L. Buttyán, and J. Hubaux, "SECTOR: secure tracking of node encounters in multi-hop wireless networks," in *ACM Workshop on SASN*, 2003, pp. 21–32.

[157] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility," vol. 19, no. 3. IEEE Press, 2011, pp. 630–643.