# Towards Trustworthy Computing on Social Networks: Measurements and New Applications

**A DISSERTATION**
**SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL**
**OF THE UNIVERSITY OF MINNESOTA**
**BY**

**Abedelaziz Mohaisen**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**
**FOR THE DEGREE OF**
**Doctor of Philosophy**

**Yongdae Kim**

**August, 2012**

# Acknowledgements

This work would not have been possible without the help of many people. First and foremost, I would like to thank my adviser, Yongdae Kim, for his constant help and endless support. Yongdae not only introduced me to the area of social networks and guided me to find interesting problems, but have also given me unlimited freedom to decide my work directions. I would like to thank Yongdae for so many hours of discussion, the random thoughts and questions that initiated much of my work, his constant doses of wisdom and advice, and for the free coffee!

I would like also to thank Nicholas Hopper for carefully reading my earlier work in my journey at Minnesota, and for giving deep and valuable feedback on my unorganized thoughts, which greatly improved my research skills. Nick is an amazing professor with deep understanding of both theory and practice of security and privacy, and working with him has been a great privilege. Both Yongdae and Nick have set an example of excellence as a researcher, mentor, instructor, and role model.

I would like to thank my thesis committee members, Professors David Knoke, Loren Terveen, and Arindam Banerjee, for their guidance through this process; their feedback has been absolutely invaluable.

I am grateful to Professor Abhishek Chandra and Professor Zhi-Li Zhang for contributing to the work reported in this dissertation, for exposing me to the areas of distributed and networked systems, and for their recommendations and help. I would like also to thank Professor John Carlis for guiding me on technical writing, and for giving me invaluable feedback on both my writing and presentation skills.

I would like to thank the current and former members of the security and cryptography research lab at the University of Minnesota for being such an instrumental part of my academic life. I would like to thank (not in any particular order) Denis Foo

Kune, Eugene Vasserman, Max Schuchard, Rob Jansen, Zi Lin, John Geddes, Chris Thompson, Huy Tran, Eric Chan-Tin, Victor Heorhiadi, Hun-Jeong Kang, Aaram Yun, Myungsun Kim, and Nayantara Mallesh. I would like to also thank my current and previous fellow graduate students in the computer science department who made my life here enjoyable; I would like to particularly thank (not in any particular order) Hesham Z. Mekky, Yanhua Li, Pengkui Luo, Esam Sharafuddin, Yingying Chen, Vijay Kumar Adhikari, Jia Zhou, Dongchul Park, Taehyun Hwang, Jaehoon Jeong, and Yu Gu.

Last but not least, I would like to thank my family in Palestine, South Korea, and the States, for their endless love and support over the years. I owe many thanks to Yosef, Adam, and Sarah for behaving themselves, and my wife, Young, for all the sacrifice she had to make by following me to Minnesota.

The work reported in this dissertation was supported in part by a research grants from the National Science Foundation (CNS-0917154), a research grant from Korea Advanced Institute of Science and Technology, and a Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota.

# Dedication

*To my family.*

## Abstract

In this work we study social networks by measurements and verification of assumption widely and blindly used for building security and communication services on top of these networks. We measure the mixing time—a major property used for building trustworthy systems on social networks, the expansion, and the betweenness in social graphs. We show that some of these properties do not hold, whereas others hold with less quality than assumed in the literature. From there, we propose to proceed to study these properties in different settings. We propose to study one of these widely properties, the mixing time, under several conditions used widely as practices for modifying social graphs: graph sampling and omission of graph directions. We explore reasons behind the quality of the mixing time, and propose several heuristics to improve it.

Motivated by the lack of work on accounting for differential trust in social network-based applications—Sybil defenses in particular, we develop four designs and use them to account for trust in a benchmark technique of Sybil defense (SybilLimit). We show empirically that our designs improve the security of this defense at some reasonable cost. We propose to extend these algorithms to other applications, Sybil defenses as well as other routing and information dissemination applications on top of social networks and rely on trust and social graphs' structure in their operation.

Finally, we use social networks for building two types of applications that make use of new properties discovered in these networks. The first application is a distributed computing service (called SocialCloud) that does not make any direct use of a global property of social graphs (such as the mixing time or the expansion). We show that nodes in SocialCloudfinish computing tasks relatively quickly even when as much as 30% of the nodes in the system are outsourcing computations. The second system we design is DynaMix, an anonymous communication service that incorporates dynamics in the social graph for anonymity. We show that dynamics of these networks can be used to improve anonymity of users.

# Contents

**Acknowledgements** **i**

**Dedication** **iii**

**Abstract** **iv**

**List of Tables** **vi**

**List of Figures** **vii**

**1 Introduction** **1**

1.1 Context and Problems . . . . . 2

1.1.1 Misuse in Content Voting Systems . . . . . 2

1.1.2 Identity in Distributed Computing Systems . . . . . 4

1.1.3 Censorship-resistant Communication . . . . . 4

1.1.4 Distributed Computing Services . . . . . 5

1.2 How Do Social Networks Help? . . . . . 6

1.3 Trustworthy Computing on Social Networks . . . . . 8

1.4 Summary of Contributions . . . . . 9

1.4.1 Measuring Social Networks Properties . . . . . 9

1.4.2 Better Assumptions by Incorporating Trust . . . . . 11

1.4.3 Building New Applications On Social Networks . . . . . 12

1.5 Roadmap . . . . . 13

**2 Measuring The Mixing Time of social Networks** **14**
2.1 Motivation . . . 14
2.2 How Graph Properties Are Used: The Mixing Time . . . 16
2.2.1 SybilLimit: Social Network-based Sybil Defense . . . 16
2.2.2 Anonymous Communication Systems . . . 17
2.3 Tools to Measure the Mixing Time . . . 18
2.3.1 Social Networks: Directed vs. Undirected . . . 19
2.3.2 The Mixing Time: The Definition . . . 20
2.3.3 The Second Largest Eigenvalue Modulus (SLEM) . . . 21
2.3.4 "Fast-Mixing" Social Graphs. . . . 21
2.3.5 The Mixing Time and Other Properties . . . 22
2.3.6 Average "Mixing Time" . . . 22
2.4 Measuring the Mixing Time . . . 22
2.4.1 Datasets . . . 23
2.4.2 Methodology . . . 23
2.4.3 Results . . . 24
2.5 Discussion . . . 29
2.6 Performance Implications—SybilLimit . . . 32
2.7 Related Work . . . 33

**3 Measuring the Mixing Time of Directed Graphs** **36**
3.1 Introduction . . . 36
3.1.1 Contributions . . . 38
3.1.2 Organization . . . 39
3.2 Related Work . . . 39
3.2.1 Measurements of The Mixing Time . . . 39
3.2.2 Systems Exploiting the Mixing Time . . . 40
3.2.3 Prior Work and Graph Preprocessing . . . 40
3.3 Directed Graphs' Mixing Time . . . 41
3.3.1 Defining Directed Graphs' Mixing Time . . . 42
3.3.2 Estimating the Stationary Distribution . . . 43
3.3.3 Determining the Proper Parameters . . . 44

3.4 Datasets and Data Preprocessing . . . . . 45
3.4.1 Datasets and Their Prior Uses . . . . . 45
3.4.2 Graphs Conversion . . . . . 47
3.4.3 Notes on Graph Conversion . . . . . 48
3.5 Results and Discussion . . . . . 48
3.5.1 Methodology . . . . . 48
3.5.2 Main Results and Discussion . . . . . 49
3.6 Implications on Applications . . . . . 54
3.6.1 Sybil Defenses . . . . . 55
3.6.2 Anonymous Communication Systems . . . . . 59
3.7 Summary . . . . . 61

**4 Measuring Other Exploited Properties in Social Networks 63**
4.1 Preliminaries and Definitions . . . . . 64
4.1.1 Betweenness . . . . . 64
4.1.2 Similarity . . . . . 65
4.1.3 Closeness . . . . . 65
4.1.4 Expansion . . . . . 65
4.2 Measuring Other Properties . . . . . 66
4.2.1 Betweenness and Sybil Attack . . . . . 66
4.2.2 Measurements of The Expansion . . . . . 66
4.3 $k$-Coreness and the Mixing Time . . . . . 70
4.3.1 The Mixing Time and $k$-Coreness . . . . . 71
4.3.2 Measurements and Results . . . . . 71
4.3.3 Improving the Mixing Time . . . . . 73
4.4 Summary . . . . . 75

**5 Incorporating Differential Trust into Social Network-based Sybil Defenses 76**
5.1 Designs to Account for Trust . . . . . 77
5.1.1 Lazy Random Walks . . . . . 77
5.1.2 Originator-biased Random Walk . . . . . 78
5.1.3 Interaction-biased Random Walk . . . . . 79

5.1.4 Similarity-biased Random Walk . . . 80
5.2 Implication of the Designs on the Mixing Time . . . 81
5.2.1 Sybil Defense Performance Over Simple Random Walks . . . 82
5.2.2 Defense Performance with Modified Walks . . . 83
5.3 All designs: A Comparative Study . . . 87
5.4 Implications of Findings . . . 87
5.5 Summary . . . 93

**6 SocialCloud: Distributed Computing on Social Networks 94**
6.1 Introduction and Preliminaries . . . 94
6.1.1 Contributions . . . 95
6.1.2 The Case for SocialCloud . . . 95
6.1.3 Social Networks and Systems Bootstrapping . . . 97
6.1.4 Economics of SocialCloud . . . 97
6.2 Use and Attack Models . . . 98
6.2.1 Use Model and Applications . . . 98
6.2.2 Attacker Model . . . 99
6.3 The Design of SocialCloud . . . 100
6.3.1 Design Options: Scheduling Entity . . . 100
6.3.2 Tasks Scheduling Policy . . . 103
6.3.3 Handling Outliers . . . 104
6.3.4 Deciding Workers Based on Resources . . . 104
6.4 Simulator of SocialCloud . . . 105
6.4.1 Flow Diagram of the Simulator . . . 105
6.4.2 Timing . . . 105
6.4.3 Settings . . . 105
6.5 Results and Measurements . . . 107
6.5.1 Social Graphs . . . 107
6.5.2 Performance When Varying the Number of Outsourcers . . . 108
6.5.3 Performance with Different Scheduling Policies . . . 110
6.5.4 Performance with Outliers Handling . . . 112
6.5.5 Performance with Variable Task Size . . . 112

6.5.6 Relationship Between Structure and Performance . . . . . 113
6.5.7 Additional Features and Limitations of Experiments . . . . . 114
6.6 Related Work . . . . . 114
6.7 Summary . . . . . 116

**7 DynaMix: Anonymity on Dynamic Social Networks 118**
7.1 Preliminaries . . . . . 119
7.1.1 System Settings and Application Scenario . . . . . 120
7.1.2 Formalization (for static graphs) . . . . . 121
7.1.3 Lower-bound on the Achieved Receiver Anonymity . . . . . 121
7.2 Dynamic Graphs for Anonymity . . . . . 122
7.2.1 Formalization: The case of Dynamic Graphs) . . . . . 123
7.2.2 Dynamic graph as a multigraph . . . . . 125
7.2.3 Dynamic graphs as weighted-graphs . . . . . 126
7.2.4 Generalized weighted graphs . . . . . 128
7.3 The Datasets . . . . . 129
7.3.1 The DBLP Dataset and its Preprocessing . . . . . 129
7.3.2 The Facebook Dataset and its Preprocessing . . . . . 131
7.4 Results . . . . . 133
7.4.1 Original unweighted graphs . . . . . 133
7.4.2 Dynamics as Weights . . . . . 133
7.4.3 Unweighted Dynamic Graphs . . . . . 135
7.4.4 Dynamics as Differential Weights . . . . . 135
7.4.5 Comparative Scenario . . . . . 139
7.5 Analysis and Discussion . . . . . 139
7.6 Related Work . . . . . 140
7.7 Summary . . . . . 142

**8 Conclusion and Future Works 144**

# List of Tables


2.1 Datasets, their properties and their second largest eigenvalues of the transition matrix . . . . . 23

3.1 The (original) datasets used for deriving directed and undirected graphs and measuring their mixing time with their statistics (number of nodes, number of edges, and the number of strongly connected components). . 46

3.2 The largest strongly connected component (SCC) of each of the different graphs in Table 3.1. Note that the percent in parenthesis of each graph correspond to the relative size of the largest SCC in the original graphs. 46

3.3 The undirected graphs resulting from converting the largest SCCs with statistics in Table 3.2. Note that the percent in parenthesis correspond to the number of added edges to make the directed graph undirected by having all edges in both directions. The percent is the number of added edges divided by *twice* the total number of edges in the resulting undirected graph. . . . . 46

3.4 The entropy and anonymity set comparison in directed and undirected graphs for random walk length $w = 6$ and for the different datasets. . . 61

4.1 Numerical results of operating Gatekeeper [**?**] on top of different social graphs with different characteristics. 10 Attackers are selected randomly and 99 distributers are sampled in each case (attack edges are $131, 145, 277$, and 344, respectively). $f$ is a security parameter, honest acceptance percent is of the whole graph size and Sybil is per attach edge. 68

5.1 Social graphs with their size, diameter, and radius. Physics 1, 2, 3 are relativity, high energy and high energy theory co-authorship respectively. 82

6.1 A comparison between the centralized and decentralized scheduler options. Compared features are resistance to failure, communication overhead, required additional hardware, and required additional trust. . . . 103
6.2 Social graphs used in our SocialCloud simulator. . . . . 108
7.1 Statistics of DBLP time-varying graphs. Metrics of comparison are number of nodes ($n$), number of edges ($m$), average clustering coefficient, diameter, and radius. . . . . 131
7.2 Statistics of Facebook time-varying graphs. Metrics of comparison are number of nodes ($n$), number of edges ($m$), average clustering coefficient, diameter, and radius. . . . . 132

# List of Figures


1.1 Rating and voting services are widely used in online marketplaces and content providers to rate their products by users. (The snapshot here is for a product on Amazon.com). . . . . 3

1.2 Censorship map of the world, where colors indicate the level of censorship. ■: No censorship, ■: Some censorship, ■: Country under surveillance, ■: Most heavily censored nations (source: Reporters Without Borders `http://en.rsf.org/`) . . . . . 5

2.1 Lower bound of the mixing time for the different data sets used in our experiments — the case of small data sets. . . . . 25

2.2 Lower bound of the mixing time for the different data sets used in our experiments — the case of large data sets . . . . . 26

2.3 Lower-bound of the mixing time compared to the mixing time when measured using the sampling method for the entire graphs brute-forcefully — different measurements meet the guarantees. . . . . 27

2.4 The commutative distribution function (CDF) of mixing time for the three physics datasets in Table 6.2. The variation distance is computed for every possible node in the graph, brute-forcefully. . . . . 28

2.5 Lower-bound vs. the top the average mixing time for a sample of 1000 nodes in each data set, where DBLP $x$ means the minimum degree in that data set is $x$. . . . . 29

2.6 Sampling vs. lower-bound measurements of the mixing time for 10K, 100K and 1000K of four large-scale datasets. . . . . 30

2.7 The average mixing time of a sample of 1000 initial distributions in several social networks using the sampling method for computing the mixing time using the definition. . . . . 31
2.8 Admission rate of SybilLimit when using different $t$. Facebook (A) and Slashdot (1) have 10,000 nodes each. . . . . 33
3.1 The (mean) mixing time of directed graphs before and after omitting directions of edges. Each of the figures corresponds to the mean of measurements of $\epsilon$ that corresponds to the given random walk length for 1000 different initial distributions of sources in each social graphs. . . . . 50
3.2 The (max) mixing time of directed graphs before and after omitting directions. Note that, by definition, these figures correspond to the mixing time of the social graph. While it bounds the mixing of all sources for which the mixing time is measured, it is less strictly representative of the quality needed for applications such Sybil defenses [**?**, **?**]. . . . . 51
3.3 The (mean) mixing time of directed graphs before and after omitting directions for short walks. . . . . 52
3.4 Acceptance rate of the honest nodes (suspects) by honest verifiers in directed and undirected graphs. Note that all undirected graphs outperform directed, except in Epinion. . . . . 55
3.5 Acceptance rate of the dishonest (Sybil) nodes by honest verifiers in directed and undirected graphs. Used walk lengths are 3, 4, 4, and 5 for Wiki-vote, Epinion, Slashdot, and Gnutella, respectively. . . . . 56
3.6 The mean entropy of random walks distributions on graphs before and after omitting directions for varying walk lengths ($H^u_w$ vs. $H^d_w$ for varying $w$ values). . . . . 57
4.1 Preliminary measurements of the betweenness of nodes in different social graphs. . . . . 67
4.2 A measured of the expansion of sets of nodes of different sizes beginning from every nodes in the given graphs as potential core for the expansion. 69
4.3 Expected expansion for various set sizes of various social graphs. . . . . 70
4.4 Core structure (slow mixing social graphs). . . . . 72
4.5 Core structure (fast mixing social graphs). . . . . 72

4.6 Mixing time measurement of Physics 1 before/after improving its mixing characteristics. . . . . 74
4.7 Mixing time measurement of Physics 2 before/after improving its mixing characteristics. . . . . 74
5.1 An illustration of the lazy random walk. For simplicity, $\alpha$ is equal for each node though it can be determined by each node locally to reflect what that node perceive as the trust of the network. . . . . 78
5.2 An illustration of the originator biased random walk. . . . . 79
5.3 The impact of the originator and lazy walks on the mixing time—(a) and (b) are for originator-biased while (c) and (d) are for lazy random walks. 84
5.4 The mixing time of four different social graphs when using simple vs. lazy, originator, and similarity-biased random walks, for each graph. While they are similar in size, a mixing time (parameterized by the same $\epsilon$) is variable. . . . . 85
5.5 The performance of SybilLimit measured for accepted honest nodes when using different lengths of lazy random walk for different social graphs. . 88
5.6 The performance of SybilLimit depends on the underlying social graph, where different graphs require different walk lengths to ensure the same number of accepted nodes. The originator-biased random walk can further influence the number of nodes accepted in each graph. . . . . 89
5.7 Accepted honest nodes in SybilLimit versus random walk length – with simple random walk. Different graphs have different quality of the algorithmic property though being with same size. . . . . 90
5.8 Accepted honest nodes in SybilLimit versus random walk length, when using the different designs to model the of trust in the social graph. The social graph of Facebook are sampled, where each has the size of 10,000 nodes. . . . . 90
5.9 Expected escaping walks per node (among 100 nodes, $r = 850$) in Facebook dataset (in Table 5.1) where $w = 6$ and $\alpha = 0.4$ for both of the originator and lazy random walks. . . . . 91

5.10 Accepted Sybil nodes over tainted tails when varying $g$in Facebook dataset (in Table 5.1) where $w = 6$ and $\alpha = 0.4$ for both of the originator and lazy random walks. . . . . 91
6.1 A depiction of the main SocialCloud paradigm as viewed by an outsourcer of computations. The different nodes in the social network act as workers for their friends, who act as potential jobs/tasks outsourcers. The links between social nodes are ideally governed by a strong trust relationship, which is the main source of trust for the constructed computing overlay. Both job outsourcers and workers have their own, and potentially different, schedulers. . . . . 101
6.2 The decentralized model of task scheduling in SocialCloud. . . . . 102
6.3 The flow diagram of SocialCloud: social graph is used for bootstrapping the computing service and recruit workers, nodes are responsible for scheduling their tasks by determining the amount of work each of its neighbors would process, and each worker (node) uses its local scheduler to determine how much time is allowed for each sub-task by its neighbors. 107
6.4 The normalized time it takes to perform outsourced computations in SocialCloud. Different graphs with different social characteristics have different performance results, where those with well-defined social structures have self-load-balancing features, in general. These measurements are taken with round-robin scheduling algorithm that uses the outlier handling policy in §6.3.3 for a fixed task size (of 1000 simulation time units). . . . . 109
6.5 The performance of SocialCloud on the different social graphs used for our experiments, demonstrating the inherent differences in the different social graphs. Both figures use $p = 0.3$ and the round robin scheduling algorithm. Left figure is when handling outliers, whereas the right figure without handling the outliers. . . . . 110

6.6 The normalized time it takes to perform outsourced computations in SocialCloud for different scheduling policies. Naming convention: U stands for unhandled outlier and B stands for handled outliers (Balanced). RRS, SFS, and LFS stand for round-robin, shortest first, and longest first scheduling. . . . . 111
6.7 The normalized time it takes to perform outsourced computations in SocialCloud, for variable task size. . . . . 113
7.1 Simple example of dynamic graph. $a_{12}^{(i)} = w(v_1^{(i)}, v_2^{(i)})$ is as per the definition. One or more of the weights $a_{12}^{(i)}$ could be zero. . . . . 124
7.2 Simple example of converting a dynamic graph (in figure 7.1) into multigraph by collapsing all images of a node to the node itself (the resulting nodes of the graph are the result of set union) and creating multiple-edges (corresponding to multiset union of the individual graphs). . . . . 125
7.3 A simple example of multigraph (shown in figure 7.2) conversion into weighted graph by summing weights of edges between every pair of nodes. 126
7.4 A toy example of the generalized weighted graph model to express dynamic graphs—the same graph in Figure 7.3 is used. . . . . 128
7.5 Degree distribution of the different snapshots of the DBLP dataset. Notice slight difference in the degree distribution across graphs. . . . . 131
7.6 Degree distribution of the different snapshots of the Facebook dataset. Notice no major change in the degree distribution across graphs. . . . . 132
7.7 Min-mean-max bar diagram of the achieved anonymity by utilizing the individual datasets of the different social graphs (snapshots) for several random walk lengths. DBLP (x) is abbreviated into DBx for space constraints. . . . . 134
7.8 The anonymity achieved (as entropy) by walking on the different *weighted* graphs in DBLP and Facebook constructed according to the dynamic method construction described earlier. Weights are linear sum of all original edge weights, and 1x indicates that the graph is constructed by computing the union graph on $G_1, \ldots, G_x$. . . . . 136

7.9 The anonymity achieved (as entropy, represented on the x-axis) by walking on the different *unweighted* graphs in DBLP and Facebook constructed according to the dynamic method construction described earlier. No weights are used and 1x indicates that the graph is constructed by computing the union graph on $G_1, \ldots, G_x$. . . . . 137

7.10 Min-mean-max bar diagram of the achieved anonymity (the entropy reflected on the x-axis) by utilizing the generalized wieghted graph model to capture dynamics; G stands for geometrical distribution, R for reciprocal, L for linear and N for no weights. . . . . 138

7.11 Average entropy for walk length of 10; all Facebook graphs (S. is single, G and R are geometrical reciprocal distributions of weights, W is weighted, and UW is unweighted, and numbers are to indicate which graph is used: 1-5 are original graphs whereas 11-15 are the dynamic graph model). . . 141

7.12 Average entropy for walk length of 10; all DBLP graphs (S. is single, G and R are geometrical reciprocal distributions of weights, W is weighted, and UW is unweighted, and numbers are to indicate which graph is used: 1-5 are original graphs whereas 11-15 are the dynamic graph model). . . 141

# Chapter 1

# Introduction

There have been many attempts in the past five years to build applications for distributed and peer-to-peer systems that exploit social networks properties. For example, social networks are used for building censorship-resistant Internet storage, content sharing and publishing, routing protocols, and Sybil defenses. In each of these applications, social networks are assumed to be well-connected and trusted. For instance, in social networks-based Sybil defenses a high quality of trust reduces the attackers' ability to produce multiple identities, thus, defending against the Sybil attack—caused by nodes with multiple identities in distributed systems. For these Sybil defenses to work, social networks are assumed to be trust possessing and fast-mixing, a formal quality of high-connectivity of these networks. Other applications require other properties, such as betweenness, expansion, well-balance, etc. Despite their importance to these applications, there has been not much efforts spent understanding the quality of these properties in social graphs and how such quality affects the performance of these designs.

To make matters worse, some common practices in the field of trustworthy computing on social networks lack rigor by claiming authentic properties in social graphs after graph manipulation, thus calling for further investigations. For example, to bring insight on their designs of trustworthy computing—Sybil defenses in particular, many researchers alter social graphs by trimming lower degree nodes, convert naturally directed graphs to undirected ones and thus undermine directionality of edges in the underlying social graph, or sample larger graphs to smaller ones that are easier to work with, among many other practices. In all of these cases, researchers pay no attention

to the altered algorithmic property in the underlying social graph and how this affects the operation of the trustworthy computing systems.

To this end, and to enable trustworthy computing on social networks, this work is focused on measuring, analyzing, and improving properties used for building applications using social networks. We proceed to this goal by large-scale *measurements* and *analyses* to understand these properties in their natural contexts, and in settings where they are altered as widely used in the literature. Mindful of lessons learned from the measurement, we then proceed to improving properties so as to improve the performance of systems built on top of social networks. Finally, we investigate the use of existing properties in social graphs, and to look at discovering other properties that can be easily achieved in variety of social networks to build trustworthy systems.

To motivate for our work and put it in the correct context, in this section we elaborate on the context and problems in distributed systems that other researchers in the community tried to solve using social networks, how and why social networks are used, and our take on these solutions. Then we proceed to describe our current findings and results, as well as our proposed work.

## 1.1 Context and Problems

Many of the social network-based computing designs proposed in the literature address limitations in cyberspace system due to the lack of trust in them. These designs rely on the trust in social networks as the main driving factor and make use of some social networks properties to enable efficient and reliable operation of cyberspace systems. In the following we review some of these systems, where social networks are heavily used in the literature to improve certain aspects of systems opration.

### 1.1.1 Misuse in Content Voting Systems

Many systems and services utilize features to improve the experience of their users. For example, marketplaces like Amazon and eBay, and online content providers like Youtube and Flickr, make use of "rating" (or voting) on goods, contents, and individuals (sellers) as means of determining their qualities—an example is shown in Figure 1.1. These ratings are ideally provided by experienced users to tell how they perceive such

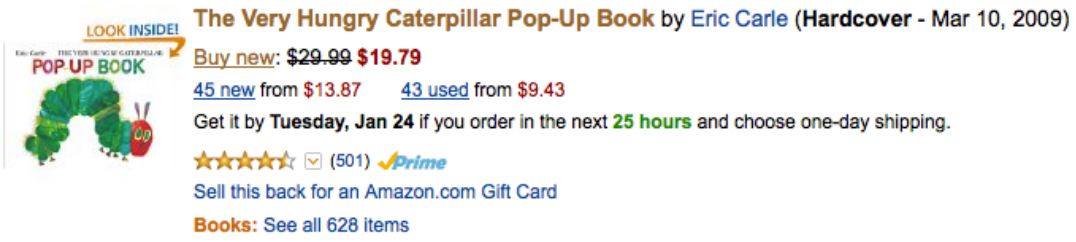



Figure 1.1: Rating and voting services are widely used in online marketplaces and content providers to rate their products by users. (The snapshot here is for a product on Amazon.com).

items, and they usually influence other new users decisions. For example, an item that is highly rated and sold by a highly rated seller on Amazon or eBay, even when it is more expensive than the same item from other sellers, is highly likely to be considered by users based on such ratings. Also, highly rated videos on Youtube are more likely to be watched based on these ratings. Likewise, items with low ratings are unlikely to be considered by new users.

In the above scenarios, users put a good faith in these ratings. They presume that such ratings come from other users who used these systems—whether it is by watching a video on Youtube or by purchasing an item on Amazon—and then casted votes to characterize their experience. These rating systems are however prone to misuse. For example, a single user may purposefully rate an item several times, thus the rating of the item may become an inaccurate representation of its quality as perceived by other users. Many of these systems require users to log-in using authentication credentials, such as an email, in order prevent them from casting multiple votes. However, most of these systems do not bind such credentials (or digital identity) with the real identity of users, thus an attacker with the intention to misuse the system will be able to create multiple email addresses and use them to cast as many votes as she or he wants.

While centralized systems, like Amazon and eBay, my prevent the problem by requesting only users who bought an item to rate it, the problem as described above is not limited to these systems. Rating (or voting) is ubiquitously used in many centralized systems—where such misuse prevention techniques may not apply; examples include aforementioned Youtube and Flickr services—and decentralized systems. For example,

ratings are used to determine the quality of videos shared on distributed file-sharing systems, where creating multiple identity to cast multiple votes or claim multiple roles in the system has proven to be easy.

The problem as described above, when a single user creates multiple identities and try to use them as she or he is multiple users, is called in the literature the "Sybil attack". Defending against the Sybil attack remains challenging in many distributed computing services.

### 1.1.2 Identity in Distributed Computing Systems

Many distributed systems rely in their functioning on the true participation and collaboration of users. Examples where collaboration include the search functionality in distributed hash tables (DHTs), routing in delay tolerant networks, content dissemination in data delivery infrastructures, among many others. The distributed nature of these systems make it hard to use a globally enforced identity, and even if such identities were to be used, the lack of centralized authority to bind these identities to real identities make it easy for a single user to create multiple identities and perform a Sybil attack, thus abuse the system.

For example, distributed hash tables (DHTs), which are widely used in distributed data management and contents distribution (e.g., files sharing), require each user to store a subset of the identities of contents or users in the system and answer queries concerning these contents or users if needed by other users in the system. Another example is collaborative computing systems (also called volunteer-based computing systems) in which users volunteer their computing resources to others to use for performing certain computations. A typical misuse scenario would include the creation of multiple identities by a single node and making the search queries fail.

### 1.1.3 Censorship-resistant Communication

One of the main threats to freedom of the online speech is censorship. Governments in most countries in the world (as shown in Figure 1.2) censor the contents on the Internet and determine beforehand what contents people can access. To circumvent censorship filters, there has been several systems but the most notable is Tor (the onion router).

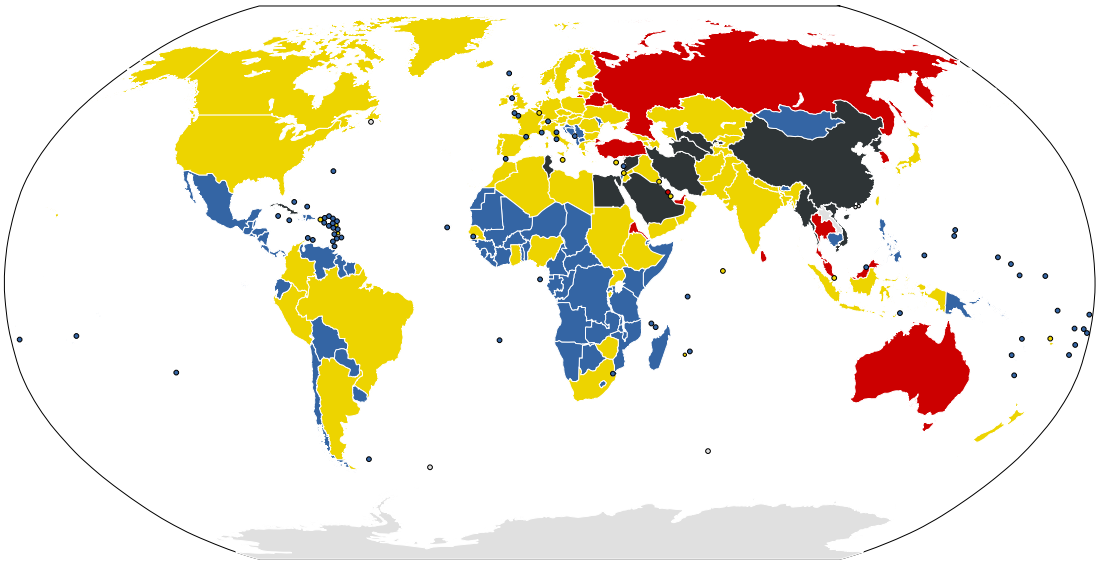


Figure 1.2: Censorship map of the world, where colors indicate the level of censorship. ■: No censorship, ■: Some censorship, ■: Country under surveillance, ■: Most heavily censored nations (source: Reporters Without Borders `http://en.rsf.org/`)

Tor uses predetermined set of volunteer servers (called relays) to mix traffic of clients so that it is hard for a traffic analyzer to trace client's activities. Two of the main limiting factors of Tor are scalability, which is addressed in several recent works as in [**?**] and [**?**], and relays blocking, which is addressed by hidden relays; also called bridges [**?**].

Another solution to the problem has been using social networks as a mixing medium to communications among people represented in these networks (details are below).

### 1.1.4 Distributed Computing Services

Many distributed computing systems rely on resource donated by volunteers, people who share similar interests for similar causes [**?**, **?**, **?**, **?**, **?**]. Compute task outsourcers put good faith in the system; they trust other volunteers in the system to perform the compute tasks as intended. Applications of such systems are particularly popular in the research community for scientific computations. One of the main issues that face this compute paradigm is the recruitment of compute workers; in order for volunteers to participate in the system, they should to convinced by the causes for which these

systems operate. This is likely to happen in academic institutes settings where institutes would be likely willing to donate their compute idle cycles for other institutes projects, given that other institutes wold do the same.

Another issue arises as the compute paradigm is made open to the public. In many of these applications, the outcome of the individual outsourced computations is usually combined to produce a result representing the different pieces of outcomes. To that end, to marginalize how misbehavior affects the operation of such systems, classical techniques like "replication" of compute tasks on different machines are used. In these systems, the same task is outsourced to different compute workers, and a voting mechanism is used for admitting results based on the outcome of the majority of the compute workers. Although that is likely to mitigate the impact of misbehavior, that could potentially be caused by users not performing computations, or generating multiple identities and taking multiple tasks with the intention of abusing the system, in many cases it is hard to reason about the trustworthiness of such systems and computations delivered by them.

## 1.2 How Do Social Networks Help?

In most of the systems and applications we described above, the origin of the problem is the lack of trust in the system. All individuals are likely to be treated equally. In a voting system, like the one used in Amazon and eBay, votes coming from supposedly different users are treated equally and presented in the system in an aggregate form to reflect qualities of products. In a distributed computing system, any node that announces availability of resources is likely to be used as a worker for any outsourced computations.

All methods proposed and used for defending against misuse and misbehavior in distributed systems try to fix this aspect in the system by bringing trust into these systems. In centralized solutions to centralized or distributed computing systems, centralized trusted authorities are used to confirm digital identities of users, match them to real identity credentials, or maintain a record of users behavior and history that can be used further to decide whether to accept or deny participation of users in the future.

Another solution to the problem looks at bringing trust from other orthogonal contexts to the distributed system where users in the distributed system are represented in that context. Accordingly, decisions whether to accept participating of users in the distributed system is tied to the trust value of these users in that orthogonal system. One good example of such contexts is social networks. Many social networks enjoy both trust characteristics and algorithmic properties, such as well-connectivity of nodes in the network to each other, that support the operation of algorithms to limit misbehavior in distributed systems. Furthermore, ties among social acquaintances in the social network can be used in a decentralized manner, without a centralized authority, making them a good fit for bringing trust into distributed and decentralized computing environment where trust is a missing factor.

Indeed, this vision has been used intensively in the past five years, where several designs are proposed in the literature to improve the operation of distributed systems using trust and algorithmic properties of social networks. Solutions using social networks included designs to defend against the Sybil attack in applications like rating and voting algorithms that are used in variety of systems, designs to improve routing in delay tolerant networks, designs to improve routing in socially selfish networks, designs to improve trustworthiness of online storage and back-up systems, designs to provide anonymity for online communications, and others to preserve user's privacy in file sharing systems, among many other systems and designs.

Most of these designs have common design principles. A typical design among those proposed in the literature would rely on the trust in the social network, and would support its operation for efficiency or security using theoretical arguments. These theoretical arguments are based on the algorithmic properties claimed in the social networks—properties that wide variety of social networks would enjoy so that the operation of these designs is possible on any of them. For example, many Sybil defenses built on top of social networks to be trust-possessing and "fast-mixing". Other Sybil defenses require social graphs to have "good" expansion, be well-balanced, or to have high betweenness of nodes so that such characteristics can be used to determine the "goodness" of nodes in the social graph and thus decide to admit or deny participations originated by these nodes in the orthogonal distributed system.

Some of these qualitative properties are assumed with certain quantities. For example, social network-based Sybil defenses using the "fast-mixing" property of social graphs assume that a short random walk with a certain length will result into certain property of the sampled nodes from the graph after a walk of that length. This is, walking on the graph for a number of steps that is in the logarithmic order of a graph size would result into sampling nodes driven from a probability distribution that is "almost identical" to the node degree distribution of the graph. Other systems assume other properties, and require them to be in certain quantities so that they work correctly in theory.

Despite the importance of these assumption for both theoretical and practical guarantees of these designs, none of the works in the literature tried to measure these properties directly in social graphs. Typically, all works in the literature inferred these properties indirectly by measuring the performance of these designs (Sybil detection, routing, information dissemination, etc). Given the experimental results of these designs on real-world social networks that meet the theoretical arguments and formulations, authors of these works have assumed that the algorithmic properties used for reasoning about the operation of these designs exist in social networks with the assumed qualities.

## 1.3 Trustworthy Computing on Social Networks

Our work in this thesis is motivated by the above arguments. We looking at bridging the gap between theory and reality in trustworthy computing at social networks. Motivated by the lack of work on measuring social networks properties used for trustworthy computing, we initiate this direction by measuring several social network properties and relate these measurements to both theoretical and practical guarantees of these designs.

Using outcomes of these measurements, we also ask and answer several natural questions: a) Do social networks have the quality of properties assumed and widely used in the literature for building trustworthy computing systems? b) If not, do these designs operate on top of social networks with the claimed guarantees even when such properties do not hold? c) Can we characterize social network properties and the reason behind their qualities? d) If so, can we improve these properties? e) Do common practices used in the literature of graph sampling and omitting graph directions affect the qualities of

these properties and the operation of these systems on top of social networks? f) Can we build new systems that either improve on prior work using reasonable assumptions or utilize new properties in social networks unused previously?

Although the main property we consider in our work is the mixing time of social graphs, which is widely used as the corner stone for building a wide variety of social network-based Sybil defenses, we also measure other properties: the expansion of graphs and betweenness of nodes that are also used for building Sybil defenses. Betweenness also has been used for improving routing in delay tolerant networks, among others. The mixing time of the graph, which is a measure of connectivity of the graph, is also used indirectly in other systems to support the efficiency of information dissemination, routing, and anonymous communication. We touch upon how these applications are affected by the qualities of these graphs.

To this end, we make two broad multifold contributions in this thesis. First, we test the underlying properties in social networks used for building trustworthy computing systems. Second, we propose several systems and protocols that either improve these properties or use new properties unseen in prior works.

## 1.4 Summary of Contributions

In this work, we make three multifold contributions. In the following, we elaborate on each of these contributions (each of these contributions represents a part of this thesis).

### 1.4.1 Measuring Social Networks Properties

Mindful of the importance of certain social networks properties for the operation of social network-based applications, such as Sybil defenses, information dissemination algorithms, and anonymous communication systems, we proceed to measure these properties in real-world social networks and graphs. Our main motivation of doing this study is as follows. First, to the best of our knowledge, some of these properties—such as the mixing time of social graphs—is not studied before on large scale social graphs. Second, it is not clear what quality of these properties is required for the operation of these applications. Whereas theoretical guarantees of Sybil defenses on social networks, for example, require certain quality of the mixing time, it is not clear if such quality

existed in social networks. Last, and as mentioned earlier, several techniques are used to alter social graphs, and it is not clear how such techniques affect the property used for building these applications. Furthermore, it is not clear how this alteration affects the operation of these applications. To this end, in this work we make the following contributions (and highlight the main interesting findings in each contribution):

1. We investigate tools and use them to measure the mixing time of undirected social graphs [**?**]. For that, we use two methods: the second largest eigenvalue modulus (SLEM) and the mathematical definition of the mixing time. We use the definition to express the richer pattern of the mixing in the social graphs. Unlike what has been claimed in the recent literature of building social network-based Sybil defenses using the "fast mixing" property, we find that many social graphs are slower mixing than anticipated and needed by such defenses. By experimenting with some of the literature defenses, we discover that the quality of the mixing time required for operating these defenses is weaker than claimed in the literature. As such, we made two firm conclusions. First, some of the theoretical guarantees claimed in these defenses based on the claimed property of the mixing time are inaccurate. Second, some of the practical security guarantees of these defenses when operated on some social graphs are weaker than claimed. Additional findings in this work show that the mixing time is greatly improved in social graphs when lower degree nodes are trimmed from the graph, a common practice in the literature.

2. While many social graphs are directed by nature, many applications are often evaluated on undirected versions of them by omitting edge directions. In this direction, we develop tools to measure the mixing time of directed graphs and develop its error bound. We then measure the mixing time of several directed benchmarking graphs and their undirected counterparts. Our initial measurements show that directed graphs are slower mixing than undirected ones [**?**]. We use two state-of-the-art applications to demonstrate the impact of these findings on designs on top of social networks: SybilLimit and "anonymity in the wild". We found that evaluation of applications on the undirected graphs always overestimates the security provided by these applications.

3. To understand why some graphs are fast-mixing and why some others are not, we related the mixing time to degeneracy, which captures cohesiveness of the graph [**?**]. We show that fast-mixing graphs have a larger single core, whereas slow mixing graphs have smaller multiple cores. We build on these observations by designing techniques to improve the mixing time of slow mixing graphs using auxiliary links [**?**]. We also show that another property recently used for building a Sybil defense—namely graph expansion, relates to the mixing time. While fast mixing graphs have good expansion, slow mixing graphs have poor expansion.

### 1.4.2 Better Assumptions by Incorporating Trust

Social trust is the other main feature used—along with the properties examined above—for building trustworthy computing applications on social networks. While most applications in the literature required certain qualities of social trust so as to reason about their operation and guarantees, the very same applications are experimented on online social graphs which are known for their weak trust characteristics.

For example, social network-based Sybil defenses do not consider the different amounts of trust represented by different graphs nor the different levels of trust between different nodes, though trust is a crucial requirement in these defenses. To address this problem, we introduced two theoretical (lazy- and originator-based) and two data-driven (similarity- and interaction-based) designs to tune the performance of Sybil defenses by accounting for differential trust [**?**].

Each of these designs biases the random walks used for operating these Sybil defenses. Interestingly, we find that the cost of operating Sybil defenses is greater in graphs with high trust than in graphs with low trust values. We discovered that this behavior is due to the community structure in high-trust graphs, requiring higher costs to traverse multiple communities. Furthermore, we showed that our proposed designs to account for trust—while they increase the cost of operating Sybil defenses on graphs with low trust value—greatly decrease the advantage of attacker.

### 1.4.3 Building New Applications On Social Networks

In parallel with other contributions, we explored the vein of social networks-based systems design. In this direction, we make three independent contributions. We build SocialCloud, a new volunteer-based time-sharing computing paradigm, DynaMix, a new anonymity enabling communication system on social structure by exploiting edge dynamics, and MeetUp, a secure encounter-based social network.

1. SocialCloud. We explore a new computing paradigm, called SocialCloud [**?**], in which computing nodes are governed by social ties driven from a trust-possessing social graph. We show that incentives to adopt this paradigm are intuitive and natural, and security guarantees provided by it are solid. We propose metrics for measuring the utility of this computing paradigm, and consider several design trade-offs for its operation. Using real-world social traces, we run an event-driven simulator of SocialCloud, and demonstrate the potential of this paradigm for ordinary users. Interestingly, we find graphs known to perform poorly for Sybil defenses [**?**] are good candidates for our SocialCloud for their "self load-balancing" features.

2. DynaMix. Existing solutions for anonymous communication on social structures undermine the impact of networks dynamics on anonymity guarantees. We propose DynaMix, an anonymous communication system that exploits dynamic structures in social networks [**?**]. We formally show an intuitive connection between anonymity on dynamic graphs and random walks on weighted graphs in which weights summarize the history of edges and allow for future dynamics to weight adjustment. We showed several measurements of our proposed model on dynamic graphs extracted from real-world social networks and compared it to static structures driven from the same graphs, highlighting potential of our proposed system enriching graph structure and improving quantitative anonymity as both entropy and anonymity sets.

## 1.5 Roadmap

The rest of this thesis is divided into three parts, each of which represents one of the contributions highlighted above. In the first part we introduce the measurements (chapter 2 through chapter 4), in the second part we introduce designs to account for trust (chapter 5), and in the third part we introduce new applications (chapter 6 and chapter 7). The thesis consists of 7 chapters, in addition to the introduction. In chapter 2 we introduce measurements of the mixing time in undirected graphs, followed by measurements of the mixing time in directed graphs, and implications on Sybil defenses and anonymous communication systems in chapter 3. In chapter 4 we measure other properties in social networks that are used for building trustworthy computing, investigate the reasons behind the quality of the mixing time in social graphs, and propose several heuristics to improve it. In chapter 5 we introduce designs to account for trust in social networks based Sybil defenses. In chapter 6, we introduce SocialCloud, a social-network based compute overlay that makes use of volunteer computing resources governed by social associations to improve trustworthiness of computations. In chapter 7 we introduce DynaMix, an anonymous communication service that makes use of network dynamics to improve user anonymity. Finally, conclusions and future works are reported in chapter 8.

# Chapter 2

# Measuring The Mixing Time of social Networks

## 2.1 Motivation

The Sybil attack is a well-known and powerful attack in distributed systems. In the basic form of this attack, a peer representing the attacker generates as many identities as she can and acts as if she is multiple peers in the system. These "virtual" peers are then utilized to influence the behavior of the system [**?**]. The number of identities that an attacker can generate depends on the attacker's resources such as bandwidth, memory, and computational power. With the sharp hardware growth—in terms of storage and processing capacities—and the popularity of broadband Internet, even attackers who use "commodity" hardware can cause a substantial harm to large systems. Classical solutions to the problem are insufficient in many distributed systems contexts, for that they assume the existence of centralized authorities in such systems [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. On the other hand, social networks-based solutions to the problem [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**] avoid any assumption on centralized authorities, which are replaced by (social) peers participation in the distributed system under certain assumptions

In these decentralized defenses, peers in the network are not merely computational entities—the human users behind them are tied to each other to construct a social network. The social network is then used for creating designs (such as those in bootstrapping the security and detecting Sybils under two assumptions: algorithmic and

sociological. The algorithmic assumption is the existence of a "sparse cut between the Sybil and non-Sybil subgraphs" in the social network, which implies a limited number of attacker edges; edges between Sybil and non-Sybil nodes. Furthermore more, these defenses assume that honest region of the social network is "***fast-mixing***"; a quantified quality of the connectivity of the honest graph (the graph that contains the honest nodes only). The sociological assumption is a constraint on the trust in the underlying social graph: the social graph used in these defenses needs to exhibit "***strong trust***" as evidenced, for example, by face-to-face interaction demonstrating social actors' knowledge of each other [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. Despite their essential role to the practicality of any potential design on top of social network, these designs are blindly accepted without rigorous verification of their validity. Other examples of scenarios of using social networks for building applications without verifying properties being used include Sybil defenses in mobile networks [**?**], the use of social networks for routing and social search [**?**, **?**, **?**, **?**, **?**, **?**], which all assume the features of "***well-balance***", "***good expansion***", "***good betweenness***", or "***clustering of graphs***", among others.

Until now (late of 2009; the time of starting this work) these assumptions are not challenged. Although several social network properties like clustering, betweenness, and connectivity properties, are widely studied and measured [**?**, **?**, **?**, **?**, **?**], none of these works is directed towards these applications built on top of social networks using these properties. To the best of our knowledge, and before our work, no work tried to measure the mixing time of social graphs and relate the quality of the mixing time in these graphs to guarantees of Sybil defenses. The only work in the literature addressing this issue is in [**?**], which does not address these defenses.

Concurrent to work, the algorithmic assumption has been indirectly questioned in [**?**], where it is shown that some Sybil defenses are sensitive to the community structure *in a selected set of social graphs*. However, the authors did not measure the mixing time nor tried to relate its quality to these systems guarantees. On the other hand, although there has been several works in the recent literature on social network infiltration [**?**, **?**, **?**]—where authors of these works have claimed that their findings would affect the operation of Sybil defenses on top of social networks, none of these works tried to quantify how these attacks would affect the operation of Sybil defenses. More importantly, none of these works tried to provide fixes for these attacks to improve the

operation of Sybil defenses.

In the rest of this chapter, we investigate measuring these properties and relate that to the guarantees of these systems. In particular, we investigate tools and use them to measure the mixing time of different social graphs and relate that to guarantees of the Sybil defenses. We propose to measure how the mixing time is affected by widely used practices, like omission of edge directions, and sampling, and how that affects two classes of applications built using this property, the Sybil defenses and anonymous communication systems. Then, we measure two other properties, namely the expansion of the graph and the betweenness of nodes. While the former is used for building a Sybil defense, the latter is used for building a Sybil defense and a routing protocol. We relate these measurements to the mixing time and applications built on top of social networks. Finally, we explore a heuristic to improve the mixing time by auxiliary links, given our understanding for the reasons beyond its quality in different graphs.

## 2.2 How Graph Properties Are Used: The Mixing Time

To show how graph properties are used for building these systems, in this section describe two of them, a Sybil defense, called SybilLimit, and an anonymous communication system, called "anonymity in the wild". Both systems rely on the mixing time for their operation, and certain qualities of the mixing time are assumed for their theoretical and practical guarantees.

### 2.2.1 SybilLimit: Social Network-based Sybil Defense

In SybilLimit, each node samples $r$ edges in the graph as "witnesses", where $r = r_0\sqrt{m}$, by running $r$ independent instances of random walks each of length $w = O(\log n)$ and picking the last edge in the walk in the sample. Under certain assumptions on the graph's mixing characteristics, there is an overwhelming probability that two sampled subsets of honest nodes/edges in the social graph will have a non-empty intersection, which would be used for suspect verification. Formally, if the social graph is fast mixing—i.e., has a mixing time of $O(\log n)$ so that the distance between the stationary distribution of

the graph and the walk graph after $O(\log n)$ is $\Theta(1/n)$ [1] —then probability of the last node/edge visited in a walk of length $O(\log n)$ drawn from the edge/node stationary distribution is at least $1 - \frac{1}{n}$ (Theorem 1 in [**?**]). Accordingly, by setting $r_0$ properly, one can use the birthday paradox to make sure that the intersection between two sampled subsets of edges (by two honest nodes) is a non-empty set with an overwhelming probability. Furthermore, given that the social graph is fast mixing, and the number of attack edges—edges that connect Sybil with honest nodes—is limited, probability for random walks originated from honest region to dishonest region are limited. The impact of such "escaping tails" on the operation of the defense is further marginalized using a "balance condition" which ensures that accepting a suspect would not cause a spike of the number of accepted suspects via a certain edge in the graph. Chances of dishonest nodes being accepted by sampling honest edges is limited, and bounded by the number of attack edges.

### 2.2.2 Anonymous Communication Systems

The idea of mixers over social links is very simple [**?**]. In these systems [**?**, **?**], users recruit their social acquaintance to relay their traffic and to provide anonymity to them. In the nutshell, each node (user) forwards her own traffic to her friends, and friends forward that traffic to their friends, and so on, for a certain number of hops, e.g. $w$. The number of hops $w$ is a system-wide parameter, which is determined by the security level desired in the system. The anonymity is defined for two parties; the sender and the receiver of traffic (we follow the same model in [**?**] for defining the anonymity of both parties).

For a sender, the anonymity defined in terms of the *anonymity set* is $n$, thus the entropy of the probability distribution of any node being the sender is $\log_2(n)$—same for both directed and undirected graphs. On the other hand, the anonymity set for a node being the receiver is determined by the probability distribution achieved after the fixed number of hops $w$ used in the system. Let the distribution of the final node selected in a *random walk* after $w$ hops be $\pi_i^w = \pi_i \mathbf{P}^w$, where $\pi_i^w = [\pi_i^w(j)]^{1\times n}$ ($\pi_i$ is an initial distribution). The anonymity of the receiver of the traffic (the last hop in the

[1] This quantity is further assumed to be $1/n$ in many lemmas in SybilLimit's proof; see for example Lemma 7 and Theorem 3 in [**?**].

walk) is measured by the entropy $H_w$, which is given as follows:

$$H_w = -\sum_{j=1}^{n} \pi_i^w(j) \log_2 \pi_i^w(j) \tag{2.1}$$

Using the entropy in Eq. (3.6), we define the *anonymity set* $A_w = 2^{H_w}$. The maximum entropy and anonymity set for a walk on a graph are achieved with the probability distribution of that walk as it approaches the stationary distribution.

We use $\overline{H_w^d}$ and $\overline{A_w^d}$ for the average entropy and anonymity sets in a directed graph, while $\overline{H_w^u}$ and $\overline{A_w^u}$ are used for the average entropy and anonymity sets in an undirected graph. We define the average entropy and anonymity sets for 1000 random walks starting from different sources (see below).

Unlike SybilLimit, where certain parameters are required for its operation and is used for its security proof, "anonymity in the wild" does not assume any qualities of the mixing time. It rather shows that a short random walk on the graph is sufficient to reach close enough to the stationary distribution so that the anonymity set of the walk is a large proportion of the maximum anonymity set (i.e., the power of the entropy in the stationary distribution).

## 2.3 Tools to Measure the Mixing Time

Theoretical results that provide tools to measure graph properties have been already studied intensively over the past decades in other contexts, including graph and probability theories. Such results can be applied directly, or with small modifications to the problem in hand, and to understand the extent to which assumptions being made about social networks exist in reality. In the following, we review some of these results and using them to understand the algorithmic properties of social graphs in real-world social networks, an assumption that is being used for reasoning about the behavior of social network based systems.

### 2.3.1 Social Networks: Directed vs. Undirected

Both directed and undirected social networks exist in many natural contexts [2] . Popular online directed social networks include Twitter, Youtube, and Google+, whereas undirected social networks include Facebook, Myspace, and LinkedIn, to mention some. While most of the literature work on using social networks to build trustworthy computing systems assumes undirected social graphs, some others (like [**?**, **?**, **?**, **?**, **?**]) have used directed graphs and altered them by either omitting edge directions entirely or by considering a connected subgraph in which an edge is established between two nodes if it is symmetric (i.e., an edge that exists in both directions). Accordingly, in this section, we consider both types of graphs, directed and undirected.

Formally, we refer to an undirected graph as $G = (V, E)$, where $V$ ($|V| = n$) is the set of nodes in the $G$ and $E$ ($|E| = m$) is the set of edges (relationships or interdependencies) between the nodes. For $G$, we define the symmetric adjacency matrix $\mathbf{A} = [a_{ij}]^{n \times n}$. where the $a_{ij} = 1$ if an edge exists between $v_i$ and $v_j$ in $V$. We define the degree of a node $v_i \in V$ as the number of nodes in $V$ adjacent to $v_i$ and denote it by $\deg(v_i)$. For $G$, we define $\mathbf{P} = [p_i j]^{n \times n}$ as the transition matrix, where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. $\mathbf{D}$ is defined as a diagonal matrix where the $ii-$th element in $\mathbf{D}$ is $\deg(v_i)$.

For clarity, we refer to a directed graph on $n$ vertices and $m$ edges as $\mathbb{G}$. Similar to the undirected graph case, let $\mathbf{A} = [a_{ij}]^{n \times n}$ be the adjacency matrix of $\mathbb{G}$, where $a_{ij} = 1$ if there is an edge from $v_i$ to $v_j$ in $\mathbb{G}$ (denoted by $v_i \to v_j$), and 0 otherwise. Let $\deg(v_i)^-$ be the out-degree of node $v_i$.

We define the transition probability matrix $\mathbf{P} = [p_{ij}]$ where $p_{ij} = 1/\deg(v_i)^-$ iff $v_i \to v_j$, and 0 otherwise. In a clean matrix form, we get

$$\mathbf{P} = (\mathbf{D}^-)^{-1}\mathbf{A}, \tag{2.2}$$

where $\mathbf{D}^-$ is a diagonal matrix in which the $ii$-th element is defined as $\sum_j a_{ij}$.

The stationary distribution, $\pi$, of random walks on $\mathbb{G}$ ($G$, respectively), is defined as a probability distribution that is invariant to $\mathbf{P}$ (i.e., $\pi = \pi\mathbf{P}$). Formally, $\pi$ is defined in Theorem 2 for undirected graphs (with certain properties; see below—the same definition also applies to the case of strongly connected directed graphs as well).

[2] In this chapter, we limit ourselves to measuring the mixing time of undirected graphs. Directed graphs are treated separately in the subsequent chapters.

**Theorem 1** *Let* $\mathbf{P}$ *be the probability transition matrix of a Markov chain that is periodic and strongly connected, defined on a graph* $\mathbb{G}$*. Then,*

$$\lim_{t\to\infty} \mathbf{P}^t = \mathbf{P}^\infty \tag{2.3}$$

*where* $\mathbf{P}^\infty$ *is an* $(n \times n)$ *matrix of identical rows, where each row equals to* $\pi$*, the stationary distribution of every walk on* $\mathbb{G}$*.*

For an undirected graph, $\pi = [\deg(v_i)/2m]^{1\times n}$, whereas $\pi$ in directed graphs in general has no closed-form expression.

### 2.3.2 The Mixing Time: The Definition

Significant fraction of social network-based designs use the mixing time as one of their assumption for operation and security guarantees. In these designs social graphs are assumed to be "fast-mixing" for all Sybil defenses. For $G$ defined in §2.3.1, recall that $\mathbf{P} = [p_{ij}]^{n\times n}$, where

$$p_{ij} = \begin{cases} \frac{1}{\deg(v_i)} & \text{if } v_i \text{ is adjacent to } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

The "event" of moving from a node to another in the graph is captured by the Markov chain which represents a random walk over the graph $G$. A random walk $R$ of length $k$ over $G$ is a sequence of vertices in $G$ beginning from an initial node $v_i$ and ending at $v_t$, the terminal node, following the transition probability defined in Eq. (2.4). The Markov chain is said to be *ergodic* if it is irreducible and aperiodic. In that case, it has a unique stationary distribution $\pi$ and the distribution after random walk of length $k$ converges to $\pi$ as $k \to \infty$ (Theorem 2). The *mixing time* of the Markov chain, $T$ is defined as the minimal length of the random walk in order to reach the stationary distribution. More precisely, $T$ (parameterized by $\epsilon$) of a Markov chain is defined as

$$T(\epsilon) = \max_i \min\{t : |\pi - \pi^{(i)}\mathbf{P}^t|_1 < \epsilon\}, \tag{2.5}$$

where $\pi$ is the stationary distribution, $\pi^{(i)}$ is the initial distribution concentrated at vertex $v_i$, $\mathbf{P}^t$ is the transition matrix after $t$ steps, and $|\cdot|_1$ is the total variation distance defined as $\frac{1}{2}\sum_j |\pi(j) - \pi_i^t(j)|$ (where $\pi_i^t = \pi_i\mathbf{P}^t$).

### 2.3.3 The Second Largest Eigenvalue Modulus (SLEM)

In addition to the definition in Eq. (2.5), which can be used to compute the mixing time of an undirected graph directly from its transition matrix, the mixing time is bounded by the second largest eigenvalue of the transition matrix $\mathbf{P}$. This is, let $\mathbf{P}$ be the transition matrix of $G$ with ergodic random walk, and $\lambda_i$ for $1 \leq i \leq n$ be the eigenvalues of $\mathbf{P}$. Then all of $\lambda_i$ are real numbers. If we label them in decreasing order, the following holds:

$$1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_{n-1} \geq \lambda_n > -1. \tag{2.6}$$

We define the second largest eigenvalue $\mu$ as

$$\mu = \max\{|\lambda_2|, |\lambda_{n-1}|\}. \tag{2.7}$$

Then, the mixing time $T(\epsilon)$ is bounded by

$$\frac{\mu}{2(1-\mu)} \log(\frac{1}{2\epsilon}) \leq T(\epsilon) \leq \frac{\log(n) + \log(\frac{1}{\epsilon})}{1-\mu}. \tag{2.8}$$

### 2.3.4 "Fast-Mixing" Social Graphs.

We say that the Markov chain is rapidly mixing if $T(\epsilon) = \text{poly}(\log n, \log \frac{1}{\epsilon})$. In literature [**?**, **?**], the rapid mixing of the Markov chain is cited as "fast mixing" for the graph [**?**, **?**, **?**, **?**]. Here, we follow the tradition of referring to this bound as "fast mixing". Also, again following these previous work, we strengthen the definition by considering only the case $\epsilon = \Theta(\frac{1}{n})$, and requiring $T(\epsilon) = O(\log n)$ [**?**, **?**, **?**, **?**]. For the mixing time definition in Eq. (2.5), and when considering an undirected unweighted graph, the stationary distribution is

$$\pi = [\pi_{v_i}]^{1\times n} = [\frac{\deg(v_i)}{2m}]^{1\times n} \tag{2.9}$$

for $i = 1 \ldots n$. As mentioned before, computing the exact stationary distribution for a directed graph in a closed-form expression is not possible, although computing an estimate and its error might be possible.

### 2.3.5 The Mixing Time and Other Properties

The mixing time is tightly related to the connectivity of the graph, which is explicitly assumed in many contexts of previous works [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]. This is, strongly-connected graphs are fast mixing and have small mixing time while weakly connected graphs are slow mixing and have large mixing time [**?**]. Some of the works cited (e.g., [**?**]) above informally refer to requiring a well-connected graph by having a expansion factor [**?**], another terms that is related to the mixing time [**?**]. Finally, the second largest eigenvalue used for measuring the mixing time (in Eq. (2.8)) bounds the graph conductance, a measure for the community structure [**?**] in these graphs. In short, the conductance is $\Phi \geq 1 - \mu$ [**?**].

### 2.3.6 Average "Mixing Time"

Although the mixing time as defined in Eq. (2.5) is defined overall random walks as the shortest walk from the worst mixing source in the graph, there has been recent work in the literature to define an average mixing time and the average conductance [**?**], rather than the worst case. This measure is further bounded by other graph properties; see definition 7.13 and Lemma 7.14 in [**?**]. In a fellow-up to our work, and agreeing with our conclusions in [**?**], the author of [**?**] suggested that the average mixing time might be in use for the Sybil defenses, rather than the definition in Eq. (2.5).

## 2.4 Measuring the Mixing Time

Now we proceed to measure the mixing time of different social graphs to see if the qualities assumed in the prior work in the literature exist in these graphs or not.

Although the mathematical tools for measuring the mixing time are known in literature, measuring the mixing time—especially of large graphs—is a computationally non-trivial task, requiring $O(tn^3)$ computations for a walk length parameter $t$ and network size $n$, and that might be the reason why fewer efforts are made to measure this essential property in large social graphs.

Table 2.1: Datasets, their properties and their second largest eigenvalues of the transition matrix

| Dataset | Nodes | Edges | $\mu$ |
|---|---|---|---|
| Wiki-vote [?] | 7,066 | 100,736 | 0.899418 |
| Enron [?] | 33,696 | 180,811 | 0.996473 |
| Physics 1 [?] | 4,158 | 13,428 | 0.998133 |
| DBLP [?] | 614,981 | 1,155,148 | 0.997494 |
| Physics 2 [?] | 11,204 | 117,649 | 0.998221 |
| Physics 3 [?] | 8,638 | 24,827 | 0.996879 |
| Facebook A [?] | 1,000,000 | 20,353,734 | 0.982477 |
| Facebook B [?] | 1,000,000 | 15,807,563 | 0.992020 |
| Livejournal A [?] | 1,000,000 | 26,151,771 | 0.999387 |
| Livejournal B [?] | 1,000,000 | 27,562,349 | 0.999695 |
| Youtube [?] | 1,134,890 | 2,987,624 | 0.997972 |

### 2.4.1 Datasets

The social graphs used in our experiments are in Table 6.2. These graphs are selected to feature two models of knowledge between nodes in the social networks. These networks are categorized as follows. (1) social networks that exhibit knowledge between nodes and are good for the trust assumptions of the Sybil defenses; e.g., physics co-authorships and DBLP. These are slow mixing, as we will see later. (2) Graphs of networks that may not require face-to-face knowledge but require interaction; e.g., Youtube and Livejournal. Closely related to those is the set of graphs that may not require prior knowledge between nodes or where the social links between nodes are less meaningful to the context of the Sybil defenses; e.g., Facebook and wiki-vote, which are shown to be fast mixing.

### 2.4.2 Methodology

Equipped with the mathematical tools explained in section 2.3, we measure the mixing time of the different social graphs shown in Table 6.2. In order to apply the tools in section 2.3 for measuring the mixing time, and to be consistent with other works in the literature of using social networks for applications that exploit the mixing time [?,

?, ?, ?, ?, ?, ?, ?], we first convert directed graphs to undirected. Our conversion method connects two nodes if an edge exists between them in either directions, or both. We further compute the largest connected component in each graph and use it as a representative social structure for measuring the mixing time, as the mixing time is undefined for disconnected graphs. For small to medium-sized graphs, we compute SLEM directly from the transition matrix of the graph. On the other hand, for feasibility reasons, we sample the representative subgraphs from each of the four large data sets (Facebook A, B and Livejournal A, B) using the breadth first search (BFS) algorithm beginning from a random node in the graph as an initial point.[3] We perform this sampling process to obtain graphs of 10K, 100K and 1000K nodes out of 3 to 5 million nodes in each original social graph. Bearing the different social graphs sizes in mind, as shown in Table 6.2, we proceed to describe the results of our experiments.

### 2.4.3 Results

Figure 2.1 and Figure 2.2 plot the lower bound of the mixing time for the different graphs in Table 6.2. We choose to use the lower-bound, but not the upper bound, because it is more relevant to the context of our study. In particular, as we observe that the lower-bound of the mixing time to satisfy a given $\epsilon$ is large, it is obvious that the mixing time for social graphs is slower than anticipated. As shown in Figure 2.1, we also observe that the mixing time is very slow, in particular for social graphs that require physical acquaintance of the social actors, as can be seen in the general tendency of these graphs. For example, physics co-authorship, Enron, and Epinion, though the social network is small, a mixing time of 200 to 400 is required to achieve $\epsilon = 0.1$. Similarly for larger social graphs, as shown in Figure 2.2, the mixing time to achieve $\epsilon = 0.1$ is varying and depends on the nature of the data set. For example, while it is about 1500 to 2500 in case of Livejournal, it ranges from 100 to about 400 in case of DBLP, Youtube, and Facebook.

To see how tight are these measurements we perform the following experiment. We first compute the lower bound of the mixing time for the physics co-authorship data

[3] Note that BFS algorithm may bias the sampled graph to have faster mixing. Since our goal is to show that the mixing time is slower than expected, this only strengthens our position. While sampling graphs to estimate certain properties is possible, as shown in [?], sampling a large graph while maintaining consistent quality of mixing time is non-trivial [?]

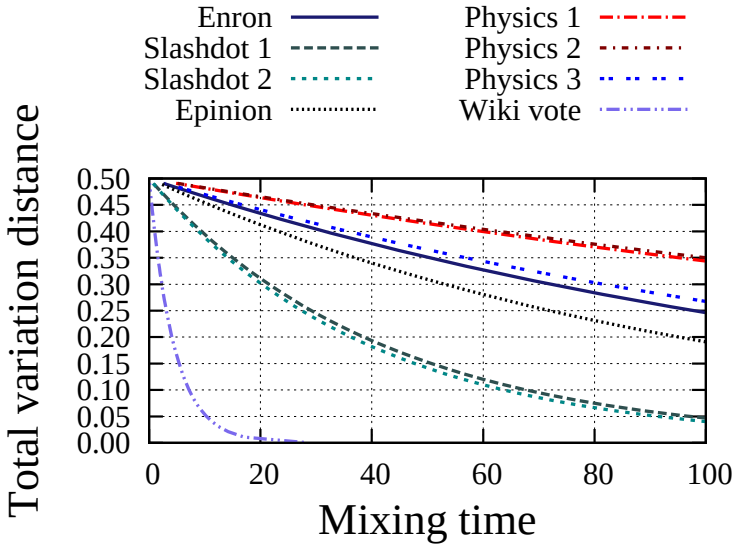



Figure 2.1: Lower bound of the mixing time for the different data sets used in our experiments — the case of small data sets.

sets, which are also reasonably small and feasible to do exhaustive computations. Then we measure the mixing time using the model in (2.5) from every possible source in the graph; the CDFs of the raw measurements are shown in Figure 2.4 for different $t$ values. We aggregate these measurements into Figure 2.3, by sorting $\epsilon$ at each $t$ and averaging values in various intervals as percentiles. We observe that while the mixing time of most sources in social graphs is better than that of the mixing time given by SLEM, the measurements using SLEM are correct since the mixing time is by definition maximum of walk lengths for given $\epsilon$ as shown in (2.5). However, even considering this effect, still for most sources the mixing time is slower than used by other papers (10 and 15 in SybilLimit).

In [**?**], Lesniewski-Laas has interpreted our results—stated in [**?**] and shown in Figure 2.3—as that walks initiated by a few sources are slow mixing while the overwhelming majority of sources have fast mixing walks. However, we notice that the slower mixing sources in some of these graphs are still large portion of nodes. For example, based on the results shown in Figure 2.4(a), even though that a random walk of length 40 yields $\epsilon < 0.3$ for about 60% of the sources in the network, the remaining sources have $\epsilon \geq 0.3$. For about 10% among the total number of nodes in the graph (about 400 nodes), $\epsilon$ for the same length of a random walk of 40 steps is about 0.5, making the

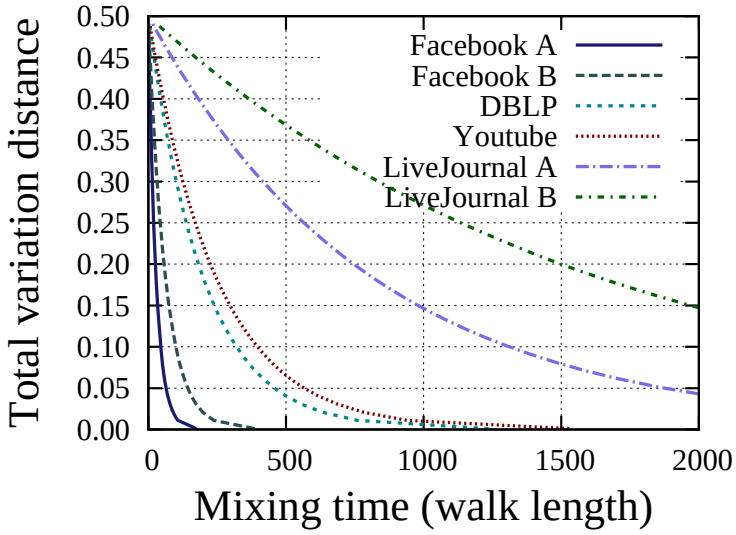
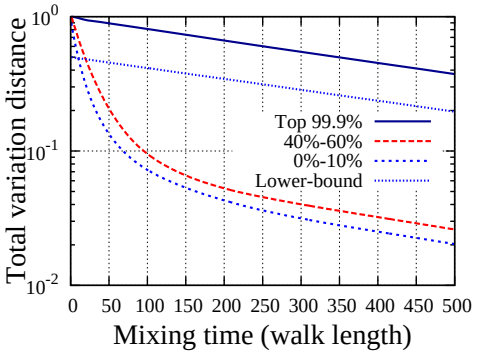


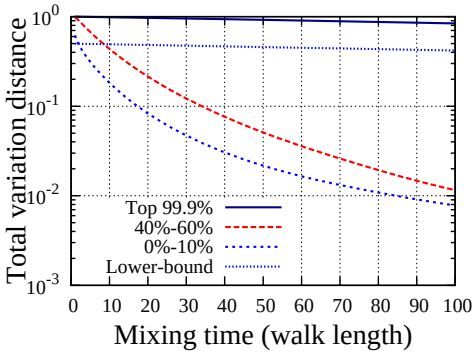

Figure 2.2: Lower bound of the mixing time for the different data sets used in our experiments — the case of large data sets

statement about the quality of "fast mixing" and "slow mixing" rather ambiguous. As we see latter, our observations in [**?**] are accurate when taken with further findings in the same context. This is, given that the quality required for operating these defenses might not be as strict as assumed, even those slower mixing sources can be considered "fast mixing" for these applications. Notice that these findings are not limited to the dataset in Figure 2.4(a) but also apply to those in Figure 2.4(b) and Figure 2.4(c). Furthermore, notice that these measurements are not in line with assumptions used in Sybil defenses, like SybilLimit, where $\epsilon$ is assumed $1/n$ for a walk length of 10 to 15.

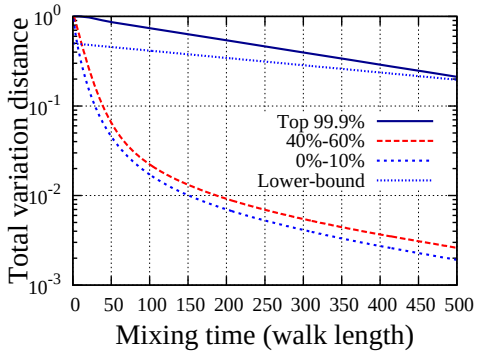
To understand the relationship between the network size and the mixing time (of the same social graph) we use the different previously sampled subgraphs, using BFS, from Facebook and Livejournal data sets (10K, 100K, and 1000K). We further measure the mixing time using SLEM and the model in (2.5) for 1000 initial distributions. We further aggregate the top 10, median 20, and lowest 10 percentile of $\epsilon$ corresponding to the given random walk, and plot them along with the mixing time derived using SLEM where the results are shown in Figure 2.6. We observe that for a million nodes graph, while the mixing time in the top 10% in the sample we computed is 100 for an averaged $\epsilon = 10^{-5}$—an excellent value to the "theoretical" guarantees of the Sybil




(a) Physics 1




(b) Physics 2



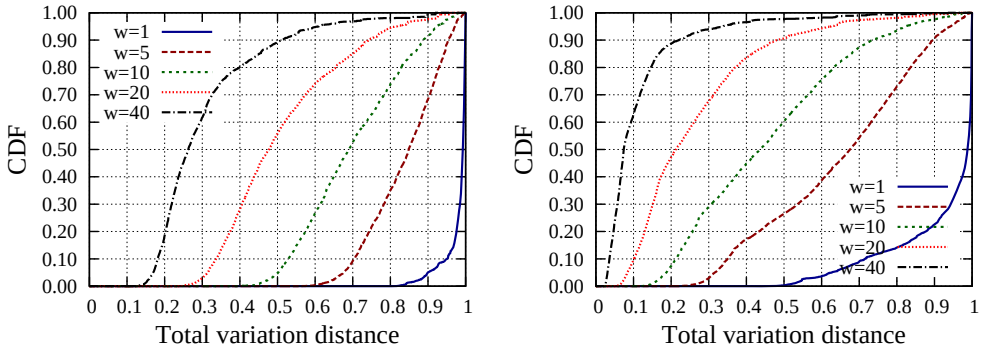

(c) Physics 3

Figure 2.3: Lower-bound of the mixing time compared to the mixing time when measured using the sampling method for the entire graphs brute-forcefully — different measurements meet the guarantees.

defenses, the SLEM-based mixing time results in only $\epsilon = 10^{-2}$ as shown in Figure 2.6(i). We attribute this difference to similar scenario as in the physics co-authorship graphs. Similar observations can be seen in each of the different large social graphs. It is worth mentioning that Livejournal (Figure 2.6(k) and Figure 2.6(l)) present poor mixing in relation with Facebook data sets, which are shown to be fast mixing.

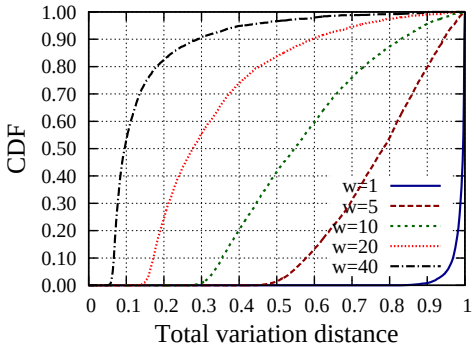
In Figure 2.7 we plot the average $\epsilon$ obtained when walking from the 1000 initial distributions as we increase $t$. We find that average $\epsilon$ is quite related to the underlying structure of graphs as well, as shown earlier for the lower bound.

Finally, to understand the methodology used for experimenting in Sybilguard and SybilLimit, we perform the same trimming technique by iteratively removing lower




(a) Physics 1

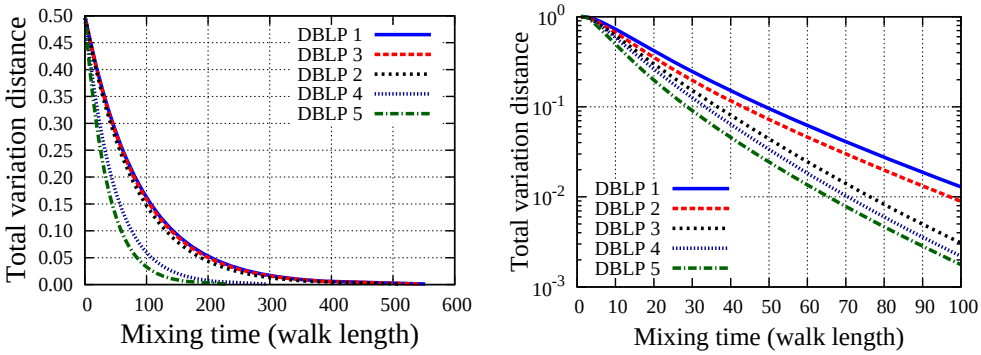
(b) Physics 2




(c) Physics 3

Figure 2.4: The commutative distribution function (CDF) of mixing time for the three physics datasets in Table 6.2. The variation distance is computed for every possible node in the graph, brute-forcefully.

degree nodes (for 1 up to 5) from the DBLP data set and computed the mixing time of the resulting graphs at each time (results shown in Figure 2.5). We observe that the pruning of lower degree greatly improves the mixing time of the social graph: for fixed mixing time of 100, by successive trimming the variation distance is reduced from about 0.2 to 0.03 (Figure 2.5(a)), and from about 0.015 to 0.002 (Figure 2.5(b)). But this improvement happens only with a huge reduction of the graph size: while DBLP 1 has 614,981 nodes, DBLP 5 has only 145,497 nodes. This means that about 75% of nodes are removed out of the social network, and could potentially be denied joining the service outright in order to boost the mixing time.




(a) Lower bound (b) Average mixing time

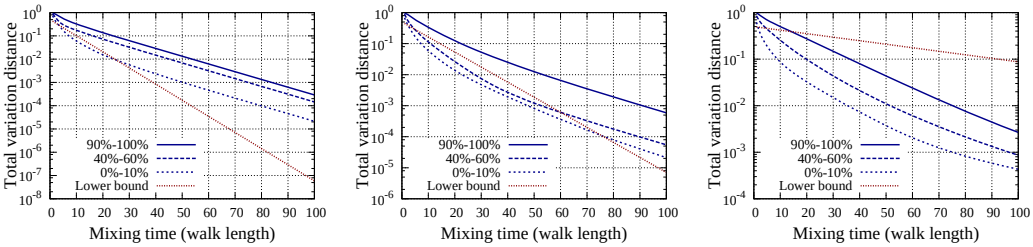
Figure 2.5: Lower-bound vs. the top the average mixing time for a sample of 1000 nodes in each data set, where DBLP $x$ means the minimum degree in that data set is $x$.

## 2.5 Discussion

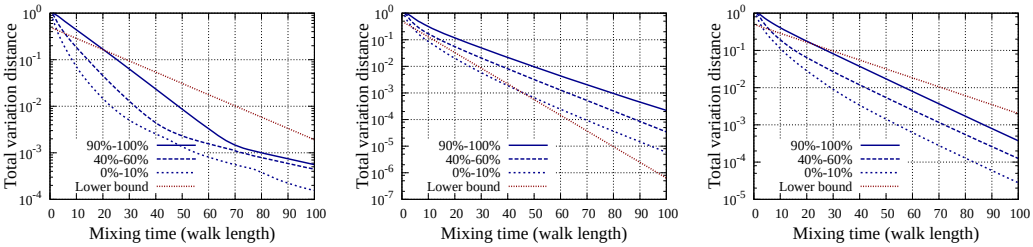
While the main finding in this study is that the mixing time of social graphs is higher than has been used in literature, we also conclude that different nodes approach the stationary distribution at different rates. This is, while the majority of walks initiated from different nodes reach closer to the stationary distribution at "higher" rate than that of the mixing time, which is defined as the maximum rate from any source, we still find—except in a few cases of online social networks—that the mixing time of the majority of nodes is larger than anticipated and used in the previous studies [**?**, **?**, **?**]. This has several implications and call for several actions.

First, since most of the theoretical guarantees of social graphs consider the model in (2.5), and since the majority of nodes in the social graphs measured in this work have better mixing time than the bound in that model, this calls for rigorous study by basing such designs and analyses on the average case of the mixing, which is relatively small, instead of the worst case of the mixing time.

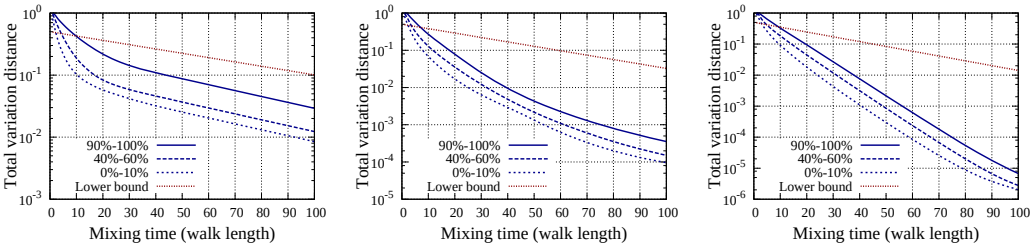
Second, the obvious implication of our findings is that one has to either give up some of the utility (service) guarantees—which are implied by that almost all honest nodes admit other honest nodes—by using relatively shorter walks, or give up part of the performance and security by enabling longer random walks in order to reach these isolated parts of the social graphs. Though this looks straightforward, going either way is not as simple as it seems. On the one hand, if one uses longer random walks in order



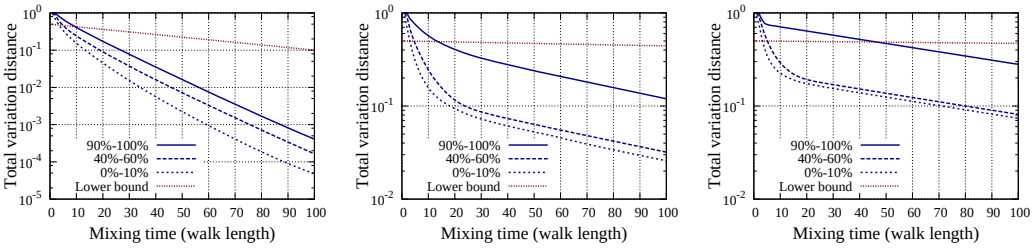

(a) Facebook A – 10K (b) Facebook B – 10K (c) Livejournal A – 10K




(d) Livejournal B – 10K (e) Facebook A – 100K (f) Facebook B – 100K

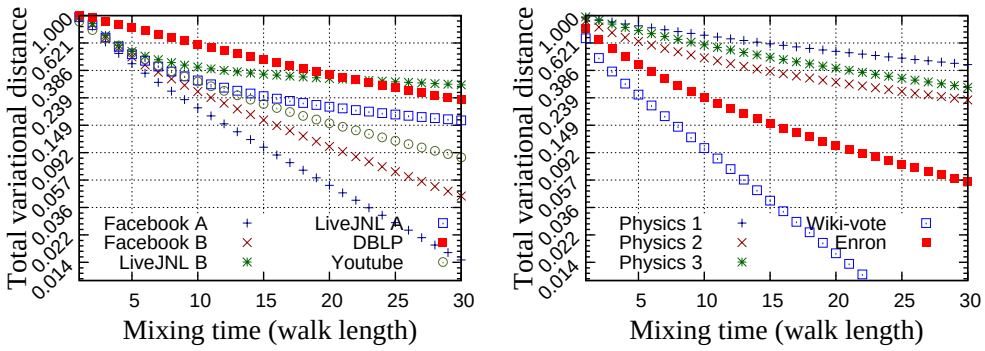



(g) Livejournal A – 100K (h) Livejournal B – 100K (i) Facebook A – 1000K




(j) Facebook B – 1000K (k) Livejournal A – 1000K (l) Livejournal B – 1000K

Figure 2.6: Sampling vs. lower-bound measurements of the mixing time for 10K, 100K and 1000K of four large-scale datasets.




(a) Large datasets

(b) Small to medium datasets

Figure 2.7: The average mixing time of a sample of 1000 initial distributions in several social networks using the sampling method for computing the mixing time using the definition.

to reach such isolated parts of the network it would be equally likely to escape to the Sbyil region which has a cut similar in its nature to that of the slower mixing part of the original social graph. On the other hand, using random walks shorter than the mixing time of the majority of nodes would also be at the expense of the utility; not only for the isolated part but also the faster mixing part as well. The end detection guarantees of the design would work as long as $g$, the number of attack edges is less than $\frac{n}{w}$.

Third, papers introducing SybilGuard, SybilLimit, and Whānau all did experiments on their schemes. Despite the short mixing time that these experiments use, their results seem to support that their schemes work as expected. The explanation of this is two part. First, the trimming of lower-degree nodes would shorten the mixing time. Second, although they claim that the social networks are fast mixing and as a part of the definition—which is also used in parts of the proofs for the theoretical guarantees—they insist $\epsilon = \Theta(1/n)$, this is a very strong burden to achieve and perhaps somewhat larger $\epsilon$ might also be good enough for these schemes to work. Also, we suspect that the difference between the average mixing time and the worst-case mixing time may have some effects on the discrepancy between the analysis and the experiment. In practice, the majority of nodes with "fast" mixing would be served and those few other nodes with very slow mixing would be denied service, which then will not be a problem for the probabilistic average-case guarantees. This last observation has been confirmed by

authors of SybilLimit in [**?**] as a potential reason why Sybil defenses would still operate reasonably although graphs are not as fast mixing as assumed.

Finally, one of the assumptions in Sybil defenses based on social networks is that the used trust model requires physical acquaintance, which is the case in social networks such DBLP and Physics co-authorship networks, for which we show slower mixing time than other "online social networks" which are known to possess less strict trust models [**?**, **?**], which by nature tolerate Sybil nodes. This calls for considering the trust model resulting from the underlying social network as a parameter, along with the mixing time, in order to evaluate the effectiveness of the social network-based defenses according to their real value. Our work in [**?**, **?**] is a preliminarily result in this direction.

## 2.6 Performance Implications—SybilLimit

In order to quantitively measure the impact of the findings of slower mixing time on the performance of Sybil defenses, we implement SybilLimit and operate it on some of the different social networks with the following settings. Since we already know the social graphs size—both $m$ and $n$, we select the proper $r$ that guarantees high probability of intersection. We set $r$ to $r_0\sqrt{m}$, where $m$ is the number of undirected edges in the graph and $r_0$ is computed from the birthday paradox to guarantee a given intersection probability. In this experiment, we consider the case without an attacker, since SybilLimit bounds the number of the Sybil identities introduced based on the number of the attacker edges. We increase $t$ until the number of accepted nodes by a trusted node (the verifier) reaches almost all honest nodes in the social network. Then, with this $t$, we find the (average) total variation distance required in each graph, which is the necessary for the operation of these designs. It is then easy to compute the number of accepted Sybil identities which is $t \times g$, where $g$ is the number of attack edges. SybilLimit works as long as $t < \frac{n}{w}$.

The result of this experiment is in Figure 2.8. We find that in some of these graphs the length of random walk is much longer than assumed previously in order to accept the majority of honest nodes. For example, even when the random walk length is 30 in Physics 1, we find that only 95% of the honest nodes are accepted, whereas the parameter $r_0$ we used ($= 4$) would ensure more than 99% acceptance rate if the graph

is fast mixing. This happens to be the case for graphs like Facebook, where more 99% of the nodes are accepted by other honest nodes for a walk length of 6.

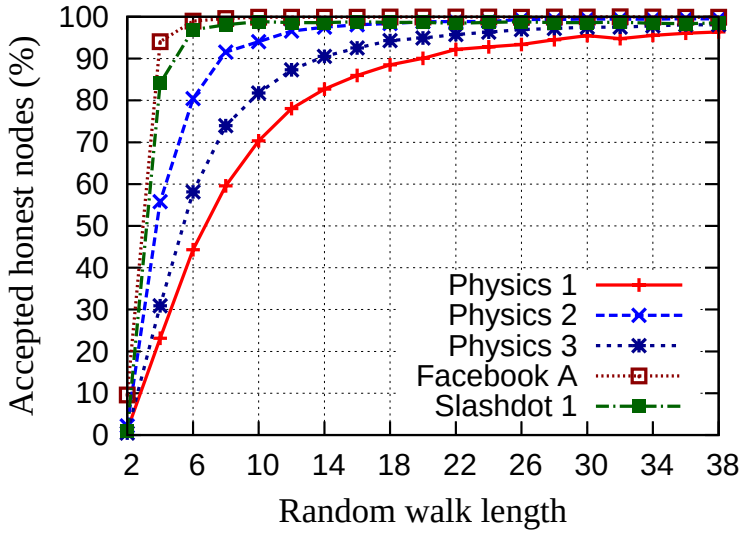



Figure 2.8: Admission rate of SybilLimit when using different $t$. Facebook (A) and Slashdot (1) have 10,000 nodes each.

## 2.7 Related Work

There has been a lot of works in the literature that try to leverage properties of social networks for applications, that along with these properties weigh the value of trust between nodes, and an algorithmic property in the social graph. In this subsection, we summarize some of these works and the properties they use.

**Sybil Defenses.** It is probably unarguable that the most popular direction of using social network has been for defending against the Sybil defenses in a decentralized manner based on the fast mixing assumption of social graphs, where every passing network or security venue has a new result in this direction [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?]. The direction has been initiated by Yu et al. [?, ?], where they used the fast mixing property of a graph to build a defense mechanism, called SybilGuard, that limits the number of Sybil identities introduced by *attack edges* (edges the connect malicious and honest nodes). In [?, ?], Yu et al. extended SybilGuard by introducing SybilLimit which further improves on the previous bounds of the number of

Sybil identities per attack edges to near optimal. An optimal solution that uses similar ingredients from SybilLimit is introduced in [?, ?] by Tran et al. All of these systems use social graphs and assume that these graphs are fast mixing according to strict mixing definition mentioned in section 2.3. Danezis and Mittal [?] used the fast mixing property to build an inference (detection) mechanism for Sybil nodes in peer-to-peer Systems. Lesniewski-Laas et al. [?, ?] introduced a routing protocol that uses the fast mixing property of the social graph that builds a distributed hash table (DHT) system. Tran et al. introduced SumUp, a Sybil-resilient online content voting [?]. Mislove et al. in [?] introduced Ostra that leverages trust in social networks and thwart unwanted communication, and indirectly mitigating the impact of the Sybil attack. Kaustz et al. introduced ReferralWeb [?], a referral system that combines social networks and collaborative filtering and assumes a well-connected social network graph, a property that is very tied to the mixing time of the graph [?]. Similar filtering systems proposed in [?, ?] as well. Despite their dependence on the assumption for their performance, none of the papers mentioned above measured the mixing time from the social networks to show that they are fast mixing, in order to meet the guarantees being theoretical proven (for more exposition see [?]).

**Anonymous communication.** A very closely related set of designs that also use the mixing time is the construction of anonymous communication systems on top of social networks. Such studies can be found in [?, ?, ?]. Other related work tries to exploit social links for privacy preserving content sharing [?]

**Betweenness to defend Sybil.** Most recently, the betweenness has been proposed by Quercia et al. in [?] as a measure for the goodness of nodes: good nodes have a higher betweenness than a threshold, when they are incorporated into their friends' networks, and bad (or Sybil) nodes have lower betweenness than a threshold. However, whether this assumption hold in reality or not is not being tested on real-world social graphs that exhibit value of trust. Our work shows that this assumption does not hold in many online and other social networks.

**Routing.** Another active field of applications that leverages social networks is routing [?, ?, ?, ?, ?, ?, ?, ?, ?]. While they are in principle similar, and most try to exploit the closeness features of a small-world social graph, these works differ in the contexts

they are proposed. For example, Davitz et al [**?**] introduces iLink, a search and routing utility on social graphs that can fit into "expert search" applications. Mabrouki et al. [**?**] exploit social networks for routing in sensor networks. Bigwood et al. [**?**] exploit social networks for routing in delay tolerant network (DTN). Chandrasekaran et al. [**?**] propose a solution for routing based on social network in P2P VoIP networks. Liben-Nowell et al. [**?**] study the potential of geographic routing in social networks. Marti et al. [**?**] study constructing a DHT-like system on top of social networks, and Sandberg [**?**] studies routing on top of social network in general contexts. Gossiping-based routing and information dissemination, on top of social networks, is studied in [**?**, **?**, **?**].

# Chapter 3

# Measuring the Mixing Time of Directed Graphs

## 3.1 Introduction

Many graphs in general, and social graphs in particular, are directed by nature. However, applications built on top of social networks, including Sybil defenses [?, ?, ?, ?, ?, ?, ?, ?], information routing and dissemination [?, ?, ?], and anonymous communication [?, ?] require mutual relationships which produce undirected graphs. When undirected graphs are used as testing tools for these applications to bring insight on their usability and potential deployment, directed graphs are converted into undirected graphs by omitting edge directions [?, ?, ?, ?] or by augmenting the underlying social graph [?, ?].

For example, Cai and Jermaine [?], Lesniewski-Laas and Kaashoek [?], Yu et al. [?], Viswanath et al. [?], Mohaisen et al. [?], and Mittal et al. [?], among others, have altered directed social graphs by either omitting edge directions entirely or by considering a connected subgraph in which an edge is established between two nodes if it is symmetric (i.e., an edge that exists in both directions). It has been claimed that the fraction of edges added by this process is small, since the majority of edges among nodes are already symmetric (e.g., see section 9.1 in [?] and section 5.1 in [?]). Also, it has been claimed that the majority of nodes remain in the largest connected component when considering edges in the undirected graph only if they are symmetric in the directed one [?, ?].

Furthermore, in most of these works it has been either explicitly argued [?, ?, ?] or implicitly assumed [?, ?] that converting a directed graph to an undirected graph will not significantly influence the graph structure nor the mixing time, the length of the random walk needed to randomly reach every node in the graph with probability proportional to the degree distribution. The quality of the mixing time has been a crucial property in these systems, where security and performance of them are based on the mixing time. In particular, most of these applications assume "fast-mixing" social graphs for their operation.

Unfortunately, it is not clear how the process of altering these graphs affects the quality of their mixing time. Although, the intuition is that directed social graphs (for which the mixing time is well-defined) would have different—and potentially slower—mixing time than undirected graphs. Motivated by the lack of prior work on this problem, we investigate mathematical tools for measuring the mixing time of directed social graphs and its associated error bounds. We use these tools to measure the mixing time of several benchmarking directed social graphs and to understand the difference in the mixing time quality between directed graphs and their undirected counterparts. We then measure how this difference impacts two applications built on top of social networks: a Sybil defense mechanism and an anonymous communication system.

The property used in both applications addressed in this chapter is the mixing time of social graphs, where these graphs are assumed to be "fast-mixing". Characterizing the mixing time by a single parameter using the second largest eigenvalue modulus (SLEM) method [?] or by the mixing time's mathematical definition, which is shown in Eq. ((3.2)), is insufficient to capture the richer mixing behavior and patterns of social graphs [?, ?]. Since this mixing behavior is the actual property utilized by many applications built on top of social networks [?], we measure it by computing the *mean* of the mixing time for walks originated from several sources in a social graph.[1] To achieve that, we use tools from the fixed point theory [?] to closely estimate the mixing time of directed graphs and its associated error bound. This estimation process is needed because, to the best of our knowledge, the literature provides no method for computing a closed-form expression of the stationary distribution of walks on directed

[1] Notice that this abuse of notation is only to simplify the exposition of our results. The reader should keep in mind that the *mixing time* is the *maximum* walk length to reach close to the stationary distribution from any source in the graph; see section 3.3 for details and ((3.2)) for the definition.

graphs—which is required for computing the mixing time.

In order to make our results relatable to other works in the literature, we experiment with directed graphs that are widely used in this context for evaluating systems built on top of social networks. By converting directed graphs to undirected ones, and measuring the mixing time of both cases, we find that undirected graphs are generally faster mixing than directed graphs. We also find that evaluation of the aforementioned systems on undirected graphs always overestimates the security of these systems.

Finally, while it might be possible to use directed graphs for building certain applications on top of social and other networks, in this work we emphasize that *we do not advocate the use of these graphs for such applications.* We rather examine a method that is widely used in the literature for modifying graphs and using them to verify the effectiveness of certain security and privacy applications. We highlight the difference in the mixing characteristics of both types of graphs, directed and undirected, and explore how this difference influences the performance and security guarantees of these applications in a "what-if" scenario. In particular, we address the following questions. What is the quality of the mixing time property in the directed graphs? What would be the performance of these applications if the directed graphs with their original property are used instead of the altered graphs? What security guarantees do these applications provide using the original property in the directed graphs before altering them? How do these guarantees compare to the guarantees claimed on the undirected counterparts?

### 3.1.1 Contributions

In this work we make two primary contributions. First, we investigate tools for measuring the mixing time of directed graphs, and use these tools to measure the mixing time of four directed social graphs, before and after graph conversion. We show the significant impact of edge directionality on the mixing time. Second, we study the impact of the quality of the mixing in these graphs, with and without edge directionality, on two applications from the literature, namely a Sybil defense and an anonymous communication system. Both applications rely on the mixing time and its quality in social graphs for their operation. In total, both of our contributions are motivated by a widely-used method in the security community for altering graphs without paying attention to how this impacts the underlying property in the original graph and security guarantees of

applications built on top of them.

### 3.1.2 Organization

The rest of this chapter is organized as follows. In section 3.2 we introduce the related work. In section 3.3, we investigate a method for efficiently computing the mixing pattern in directed graphs, and a method for bounding the error due to estimating the stationary distribution of such graphs. In section 3.4 we review the datasets and the method used for their preprocessing. In section 3.5, we outline the first part of our findings by measuring the mixing of several social graphs, before and after omitting directionality of their edges. In section 3.6, we investigate the impact of the difference in the mixing time on Sybil defenses and anonymous communication systems. Our concluding remarks appear in section 3.7.

## 3.2 Related Work

To the best of our knowledge, there is no prior work on measuring the mixing time of directed graphs. Although, several works measure the mixing time in undirected graphs generated by omitting directions in naturally directed graphs [**?**, **?**]. As such, there is no prior work on examining how the difference in the mixing time in naturally directed graphs and their undirected counterparts affects the performance of social network-based designs. On the other hand, there has been several works on designing systems on top of social graphs that exploit the mixing characteristics and other social network properties. Some of these works alter directed graphs to undirected ones using the methods mentioned in section 3.1. In the following, we review some of these works.

### 3.2.1 Measurements of The Mixing Time

The "fast-mixing" property of social graphs has been utilized for building social network-based Sybil defenses and for reasoning about their security guarantees. However, none of the authors of these defenses considered measuring the mixing time in social graphs directly. To address this issue, Mohaisen et al. [**?**] and Dellamico et al. [**?**] measured the mixing time of several social graphs. Mohaisen et al. [**?**] neglected edge directions entirely by following prior works which argued or assumed that altering graphs in that

way will not affect social graphs and their properties [?, ?, ?]. Accordingly, all social graphs including directed ones are considered undirected [?]. Dellamico et al. [?] measured the mixing time of four large social graphs, and while they used some directed graphs it is unclear what techniques they used for computing their mixing time; suggesting that a conversion of the graph as in other works [?, ?, ?] was likely performed before measuring the mixing time. Extensions of tools to measure the mixing time to directed graphs have been shown in [?].

### 3.2.2 Systems Exploiting the Mixing Time

The significance of the mixing time of social graphs is because of two categories of applications that use it for their operation: Sybil defenses, and anonymous communication Systems. Sybil defenses that utilize social networks include SybilGuard [?], SybilLimit [?], SybilInfer [?], SumUp [?], GateKeeper [?], Whanau [?], and X-Vine [?], among others—a summary and comparisons of most of these works can be found in the recent work of Haifeng Yu [?]. All of these defenses require social networks to be "fast-mixing", informally meaning that a short random walk originated from any honest node in a social graph is sufficient to reach every honest node in the graph with a probability proportional to the node's degree. The fast mixing property is required in all of these designs for both performance and security reasons [?].

Anonymous communication systems on top of social networks include "anonymity in the wild" [?] and Drac [?]. In section 3.6 we elaborate on "anonymity in the wild" [?] and a Sybil defense (called SybilLimit [?]), which we use to demonstrate our results and findings.

### 3.2.3 Prior Work and Graph Preprocessing

The pre-processing methods of directed graphs in the prior literature motivate this work. While some directed social graphs are used to demonstrate Sybil defenses in many of the above works, they are converted to undirected graphs by entirely neglecting edge directions [?, ?, ?] or by considering largest connected component with symmetric edges only [?, ?]. Each method alters the social graphs differently. While the first method is likely to add edges to guarantee symmetry of edges in the resulting graph, the second

method is likely to trim nodes to ensure that no asymmetric edges are in the resulting graph—trimmed nodes are connected to other nodes in the graph with asymmetric edges.

We observe that both methods may result in a well mixing graph from a graph that violates the mathematical definition of the mixing time (i.e., a graph for which no mixing time is defined; see section 3.3 for more details on the mathematical definitions). In both methods, the original directed graph might be weakly connected for which no mixing time is defined, yet adding additional edges or removing nodes would make it mix well. This observation has two implications. First, comparing the mixing time of graphs after processing them to the mixing time of the original graph using either of the two methods might not be possible. Second, by violating the mathematical definition the ultimate result might be obvious and uninteresting: certain nodes in a weakly connected graph will never be visited and the mixing time of the directed graph is likely to be larger than the modified graphs.

To this end, we limit our discussion to the case where the largest strongly connected component (SCC) is used to represent the directed graph. We note that the largest SCC of a graph includes the graph produced using the second pre-processing method. Furthermore, we note that a comparison between directed and undirected graphs produced using the first method only makes sense when the original directed graph has the SCC property. We mainly use the first method of pre-processing, and lightly discuss the second method, when comparing directed and undirected graphs in terms of mixing and performance of applications on social networks.

## 3.3 Directed Graphs' Mixing Time

Informally, conditions required for defining the mixing time in undirected and directed social graphs are quite similar. Random walks on a graph have a stationary distribution that is used for bounding these walks' distribution if the graph is connected. In order for the mixing time to be well defined on a directed graph, it needs to be a *strongly connected component* (SCC). In an SCC, there exists a path between every pair of nodes. Computing the SCCs of a graph in linear time is done using Tarjan's algorithm [**?**], which we use in this work. We use the *largest* SCC (in size) as a representation of each directed

graph and measure its mixing time (see justification in section 3.2). In this section, we present theoretical tools we need for measuring the mixing time of directed graphs. We define the mixing time for directed and undirected graphs in section 3.3.1, estimate the stationary distribution in section 3.3.2, and bound the error in our measurements due to the stationary distribution estimation in section 3.3.3.

### 3.3.1 Defining Directed Graphs' Mixing Time

Let $\mathbb{G} = (V, E)$ be a directed and strongly connected graph, where $|V| = n$ and $|E| = m$. Let $\mathbf{A} = [a_{ij}]^{n \times n}$ be the adjacency matrix of $\mathbb{G}$, where $a_{ij} = 1$ if there is an edge from node $v_i$ to node $v_j$ in $\mathbb{G}$ (denoted by $v_i \to v_j$), and 0 otherwise. Let $\deg(v_i)^-$ be the out-degree of node $v_i$. We define the transition probability matrix $\mathbf{P} = [p_{ij}]$ where $p_{ij} = 1/\deg(v_i)^-$ iff $v_i \to v_j$, and 0 otherwise. In a clean matrix form, $\mathbf{P} = (\mathbf{D}^-)^{-1}\mathbf{A}$, where $\mathbf{D}^-$ is a diagonal matrix in which the $ii$-th element is defined as $\sum_j a_{ij}$. The stationary distribution, $\pi$, of random walks on $\mathbb{G}$, is defined formally as a distribution that is invariant to transition probability. That is, $\pi = \pi\mathbf{P}$. Unlike the stationary distribution of an undirected graph, which is characterized in a closed form in terms of the degree distribution (This is, $\pi = [\deg(v_i)/2m]^{1 \times n}$ for $1 \leq i \leq n$), the stationary distribution in directed graphs has no closed-form expression.

Formally, $\pi$ is defined in Theorem 2 (which also applies to the case of undirected graphs as well).

**Theorem 2** *Let* $\mathbf{P}$ *be the probability transition matrix of a Markov chain that is aperiodic and strongly connected, defined on a graph* $\mathbb{G}$*. Then,*

$$\lim_{t \to \infty} \mathbf{P}^t = \mathbf{P}^\infty \tag{3.1}$$

*where* $\mathbf{P}^\infty$ *in* ((3.1)) *is defined as an* $(n \times n)$ *matrix of identical rows, where each row equals to* $\pi$*, the stationary distribution of every walk on* $\mathbb{G}$*.*

Same as in undirected graphs, the mixing time of a directed graph $\mathbb{G}$ is defined as:

$$T(\epsilon) = \max_i \min\{t : |\pi - \pi_i \mathbf{P}^t|_1 < \epsilon\}, \tag{3.2}$$

where $\pi_i$ is the delta distribution of 1 concentrated at the $i$-th position in a $(1 \times n)$ probability vector representing the probability distribution when starting a walk from

node $v_i$, and $|\cdot|_1$ is the total variation distance defined as $\frac{1}{2}\sum_j |\pi(j) - \pi_i^t(j)|$ (where $\pi_i^t = \pi_i \mathbf{P}^t$). Notice that the mixing time of the entire graph, as defined in ((3.2)), is the longest walk starting from the worst initial distribution to reach the worst reachable part of the graph [**?**] and to achieve a fixed total variation distance from the stationary distribution $\pi$.

While the mathematical definition in Eq. ((3.2)) is necessary to characterize the mixing time of the entire graph, richer patterns of mixing are expressed by the distribution of the mixing time obtained after $t$-step walks starting from different initial distribution in the graph. This distribution can be further characterized by the mean or median $\epsilon$ at a given walk length $t$. Both of the mean and median of the mixing time are usually far from the "worst-case scenario" expressed in Eq. ((3.2)) [**?**], and are usually representative to the quality of the mixing property required for the operation of Sybil defenses [**?**]. In this work, we are interested in both characterizations of the mixing time: the distribution (mean and median) and the worst case as per the definition (maximum).

### 3.3.2 Estimating the Stationary Distribution

Computing $\pi$ as in Eq. ((3.1)) is only theoretically possible, since it requires computing $\mathbf{P}^t$, where $t \to \infty$. Even when setting $t$ to some large number that is an order of magnitude of the size of the graph (e.g., $10^6$ for 1000 nodes graph), the process becomes both time and space inefficient for typical social graphs due to their large size. In part, this inefficiency is due to losing sparsity of $\mathbf{P}$ as $t$ grows thus requiring multiplication and storage of a non-sparse large matrix ($\mathbf{P}^t$). This inefficiency turns into infeasibility as the size of the graph grows to a few thousands of nodes, which is smaller than benchmarking directed graphs used in this study. Accordingly, other methods are required for computing $\pi$, or a good estimate of it. In the following we devise such method without modifying the structure of $\mathbf{P}$.

We observe that regardless of the initial distribution, every walk on a strongly connected graph ultimately converges to the stationary distribution $\pi$. We could further make the total variation distance between the ideal (unknown) stationary distribution and the distribution of a random walk beginning from an arbitrary node in the graph

arbitrarily small. In other words, given an arbitrary initial distribution $\pi_i$ and the transition matrix $\mathbf{P}$ of a strongly connected graph, we can compute $\pi_i^\ell = \pi_i \mathbf{P}^\ell$ such that $|\pi_i^\ell - \pi|_1 < \delta$, where $\delta$ is close to 0, for some large walk length $\ell$.

The convergence of $\pi_i^\ell$ to $\pi$ is a guaranteed property of aperiodic Markov chains on connected graphs [**?**]. Without knowing $\pi$, one can use $\pi' = \pi_i^\ell$ as an estimate for $\pi$, for some large $\ell$. Such $\pi'$ is sufficient to measure a very close estimate of the mixing time of directed graphs, and to serve the purpose of our measurement in understanding the difference between the mixing patterns in directed and undirected graphs. Notice that computing $\pi_i^\ell = \pi_i \mathbf{P}^\ell$ does not require computing $\mathbf{P}^\ell$. We can iteratively compute $\pi_i^\ell$ using a vector-matrix multiplications of $\pi_i$ and $\mathbf{P}$ by observing that $\pi_i^\ell = \pi_i^{\ell-1}\mathbf{P}$, where we can benefit from the sparsity of $\mathbf{P}$. In our measurements, we set $\pi_i$ to a uniform distribution (i.e., $\pi_i = [1/n]^{1\times n}$) and $\ell$ to a large number that makes error in our measurements arbitrarily small. In the following subsection, we elaborate on the method we use for setting $\ell$.

### 3.3.3 Determining the Proper Parameters

While one can argue that setting $\ell$ to a large number would make the distribution of a random walk of length $\ell$ starting from an arbitrary distribution close to the stationary distribution—and thus the former distribution can be used as an estimate of the latter distribution, the distance between both distributions is required to identify an upper bound on the error in the measured mixing time. Fortunately, the fixed point theory [**?**] provides tools to estimate the required length of the walk $\ell$ in order to achieve a certain distance from the stationary distribution (also known as the "fixed point" distribution). The following definition and theorem outline the main results needed for bounding the distance between the estimated stationary distribution, after $\ell$ iterations, and the ideal stationary distribution.

**Definition 1 (Contraction mapping [?])** *Let $(X, d)$ be a non-empty and complete metric space. A mapping $T : X \to X$ is said to be a contraction mapping if there exists a real number $\lambda < 1$ such that $d(T(\pi_x), T(\pi_y)) \leq \lambda d(\pi_x, \pi_y)$ for all $\pi_x, \pi_y \in X$ (where $d(.)$ is a metric, or distance, defined on points in the space $X$; e.g., $|\cdot|_1$).*

**Theorem 3 (Fixed point theorem [?])** *Let $(X, d)$ be a non-empty and complete metric space and let $T$ be a contraction mapping defined on $X$. For that mapping, there exists a fixed point $\pi$ s.t. $T(\pi) = \pi$. To find the fixed point, an iterative mapping of an initial point $\pi_i$ would result in a sequence of points $\pi_i^0, \pi_i^1, \pi_i^2 ... \in X$, where the following inequality describes their convergence rate to the fixed point:*

$$d(\pi, \pi_i^{w+1}) \leq [\lambda/(1-\lambda)]d(\pi_i^{w+1}, \pi_i^w) \tag{3.3}$$

Using Theorem 3 and by recalling that the matrix multiplication, which represents random walks on the graph, is a contraction mapping [**?**], the following inequality characterizes the walks convergence rate to the stationary distribution (at length $w+1$):

$$|\pi - \pi_i \mathbf{P}^{w+1}|_1 = \delta \leq [\mu/(1-\mu)]|\pi_i \mathbf{P}^{w+1} - \pi_i \mathbf{P}^w|_1, \tag{3.4}$$

where $\mu$ is the second largest eigenvalue of the transition matrix $\mathbf{P}$. By setting $\ell = t+1$, for large $t$, we make $\frac{\mu}{1-\mu}|\pi_i \mathbf{P}^\ell - \pi_i \mathbf{P}^{\ell-1}|_1$ arbitrarily small, and thus make the distance between $\pi$ and $\pi' = \pi_i \mathbf{P}^\ell$ within an acceptable range that does not influence the measured mixing time. Using ((3.4)), we bound the error in measurements starting from any node $v_j$ after $t$ steps (using the triangular inequality) as:

$$|\pi - \pi_j \mathbf{P}^t|_1 \leq |\pi' - \pi_j \mathbf{P}^t|_1 + |\pi' - \pi|_1 \tag{3.5}$$

Eq. ((3.5)) tells that we can bound the error in the measured mixing time using the estimated stationary distribution ($\pi'$) and the bounded distance between it and the stationary distribution in Eq. ((3.4)). Since we can make the right hand side in Eq. ((3.4)) arbitrarily small (we make it $< 10^{-15}$ in our measurements), and since we are interested in a region where the computed distance is within $10^{-1}$ to $10^{-12}$; c.f. section 3.5, the error due to estimating the stationary distribution is considered negligible.

## 3.4 Datasets and Data Preprocessing

### 3.4.1 Datasets and Their Prior Uses

We use four datasets in this work as shown in Table 3.1. Three of the four datasets represent social networks, whereas the fourth represents relationships in a peer-to-peer

Table 3.1: The (original) datasets used for deriving directed and undirected graphs and measuring their mixing time with their statistics (number of nodes, number of edges, and the number of strongly connected components).

| **Dataset** | **# nodes** | **# edges** | **# SCC** |
|---|---|---|---|
| Slashdot [**?**] | 77,360 | 905,468 | 6,724 |
| Epinion [**?**] | 75,879 | 508,837 | 42,176 |
| Wiki-vote [**?**] | 7,115 | 103,689 | 5,816 |
| Gnutella [**?**] | 10,876 | 39,994 | 6,560 |

Table 3.2: The largest strongly connected component (SCC) of each of the different graphs in Table 3.1. Note that the percent in parenthesis of each graph correspond to the relative size of the largest SCC in the original graphs.

| **Dataset** | **# nodes (%)** | **# edges (%)** |
|---|---|---|
| Slashdot | 70,355 (90.94) | 818,310 (90.37) |
| Epinion | 32,223 (42.47) | 443,506 (87.16) |
| Wiki-vote | 1,300 (18.27) | 39,456 (38.05) |
| Gnutella | 4,317 (39.69) | 18,742 (46.86) |

Table 3.3: The undirected graphs resulting from converting the largest SCCs with statistics in Table 3.2. Note that the percent in parenthesis correspond to the number of added edges to make the directed graph undirected by having all edges in both directions. The percent is the number of added edges divided by *twice* the total number of edges in the resulting undirected graph.

| **Dataset** | **# nodes** | **# edges** | **Added edges (%)** |
|---|---|---|---|
| Slashdot | 70,355 | 459,620 | 100,930 (10.98) |
| Epinion | 32,223 | 342,013 | 240,520 (35.16) |
| Wiki-vote | 1,300 | 36,529 | 33,602 (45.99) |
| Gnutella | 4,317 | 18,742 | 18,742 (50.00) |

file sharing system. In Slashdot Zoo [**?**], a directed edge between two nodes indicates that the first node tags the second node as a friend. In Epinion [**?**], a "who-trust-whom" online social network, an edge between two nodes indicates that the first node has tagged the second node as a trusted node. In wiki-vote [**?**], a link between two nodes indicates that the first node has voted for the second node. In Gnutella [**?**] an edge between two nodes indicates that is first node (host) is connected to second one. Notice that we do not advocate the use of these (interaction) graphs for building Sybil defenses and anonymous communication systems. We use the fact that these graphs are already benchmarks for testing such applications to validate our results based on them.

We note that three of these datasets (Wiki-vote, Epinion, and Slashdot) were previously used for measuring the mixing time [**?**], two (Wiki-vote and Gnutella) were used for demonstrating the efficiency of a social network-based Sybil defense [**?**], and one (Wiki-vote) was used for analyzing social network-based Sybil defenses [**?**].

### 3.4.2 Graphs Conversion

In each directed graph, we compute the largest SCC in order to satisfy the connectivity condition required for measuring the mixing time. The SCC of each graph and its relative size compared to the original graph are shown in Table 3.2. The largest SCC varies in size, and ranges from as low as 18% of total nodes in the original graph (38% of edges, as in Wiki-vote) to as high as 90% of nodes (and edges, as in Slashdot). For each graph, we compute $\pi'$ as in section 3.3.3, and make the distance to $\pi$ a negligible factor.

We convert each SCC to an undirected graph. In each SCC we first exclude self-loops, if any existed. Then, given the adjacency matrix of the self-loops-free SCC, $\mathbf{A}$, we compute the adjacency matrix of the corresponding undirected graph as $\mathbf{A} \vee \mathbf{A}^T$, where $\mathbf{A}^T$ is the transpose of $\mathbf{A}$ and $\vee$ is an bitwise logical *or* operation. The resulting graphs and their statistics are shown in Table 3.3. While the graphs before and after conversion maintain the same number of nodes, some additional (directed) edges are added to create edge-symmetry and to produce undirected graphs. The number of the added edges in each of the undirected graphs as a percent of the total number of edges ranges from as low as 11%, as in Slashdot, to as high as 50%, as in Gnutella and Wiki-vote.

### 3.4.3 Notes on Graph Conversion

We emphasize several aspects of the method used for graph conversion in this work. First, we note that recent works on social network-based Sybil defenses [**?**, **?**, **?**, **?**, **?**] have altered social graphs in several ways, some which are similar to what we used in this work (see section 3.2 for more details). These works and the methods used in them for altering graphs are the main motivation of this work. Second, obtaining the largest SCC from the directed graph as used this work is for a mathematical *necessity*, despite potentially trimming a portion of the original directed graph (see Table 3.1 for details). Third, our method—by first obtaining the largest SCC and then converting the directed graph to undirected—is *milder* than both literature methods explained in section 3.2.

Compared to the first method in the literature [**?**, **?**, **?**], our method ensures that the mixing is well-defined on both graph types, directed and undirected. Compared to the second method [**?**, **?**], our method trims less nodes in the directed graph, and still brings insight on the difference between both graph types when used for these applications.

We note that prior work [**?**] suggests that the majority of nodes belong to the SCC in large-scale social graphs. Thus, our findings may apply to these graphs when used by omitting directions according to the first method as well.

## 3.5 Results and Discussion

Now we proceed to measure the mixing time of directed graphs shown in Table 3.2 and undirected graphs shown in Table 3.3. We consider each graph with and without directions (as explained above) and compute the stationary distribution appropriately. For directed graphs we compute the stationary distribution using the method in section 3.3.2 and for undirected graphs we compute it using the method in section 3.3.1.

### 3.5.1 Methodology

For each graph in tables 3.2 and 3.3, we measure $\epsilon$—the total variation distance between the stationary distribution and the accumulated distribution—after $w$ steps. We initially set $w$ to 1 and increase it for a fixed to 50, with steps of 1. A walk length of 50 is sufficient to characterize the mixing rate of walks on all graphs we used. Using these

measurements, we then restrict the walk length to 10. A walk of up to that length is used in the literature for demonstrating the operation of applications of top of social networks.

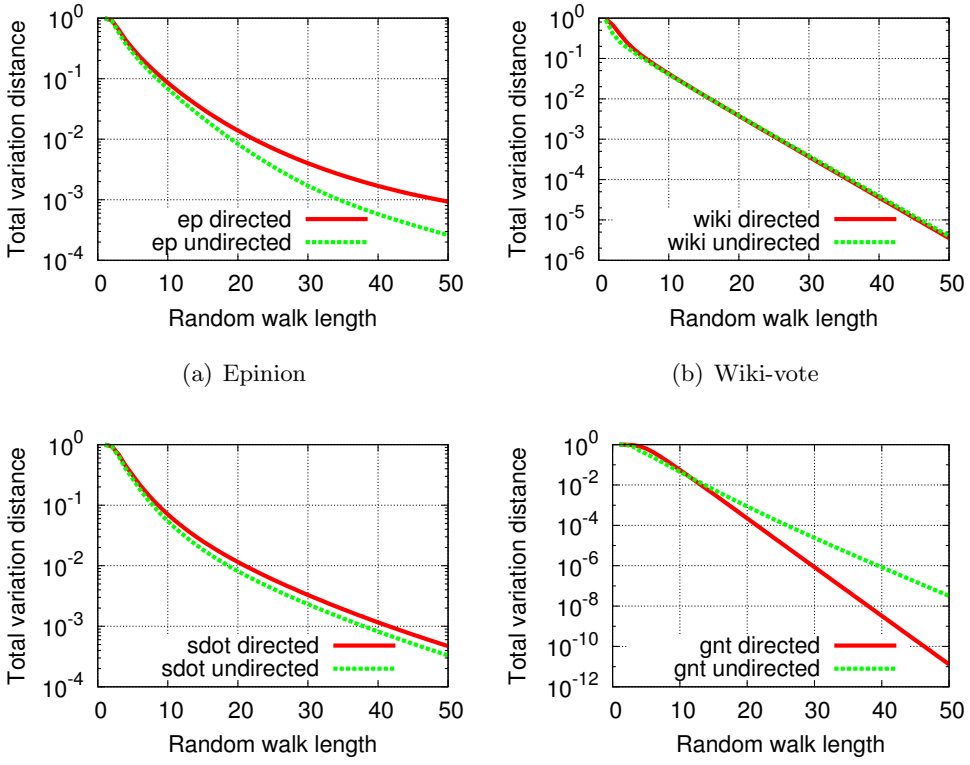
For each graph, we repeat this process by beginning from 1000 different nodes as sources of initial distributions in order to capture the pattern of mixing in these graphs. Finally, to eliminate bias caused by starting from different initial distributions, we fix each of the 1000 nodes (labels) used for both measurements. We first randomly select 1000 nodes for experimenting with the directed graphs, and use them again for estimating the mixing time of the undirected graph, after graph conversion.

### 3.5.2 Main Results and Discussion

The main results of measuring the mixing time are shown in Figure 3.1 and Figure 3.2. Figure 3.1 plots a comparison of the *mean* mixing characteristics between graphs before and after modifications. We plot the mean $\epsilon$ at each walk length, as we increase the random walk length from 1 to 50. The mean is computed over all $\epsilon$'s obtained by starting from each of the 1000 initial distributions (nodes). Figure 3.2 shows the maximum $\epsilon$ as we increase $t$, making $t$ the mixing time for the computed $\epsilon$ by definition (see section 3.3.1).

First, we compare the mixing characteristics across different datasets. We notice that different graphs, and regardless to edge directions, have different mixing characters. For example, whereas slashdot and Epinion have similar mixing pattern, Wiki-vote mixes faster than both of them. Furthermore, we notice that Gnutella mixes faster than all of the three other graphs, regardless to edge direction.

While some of the difference in the mixing characteristics across different datasets is attributed to graph density and size, as shown in Table 3.2 and Table 3.3, other crucial factor of determining the mixing time of these graphs is their structure. For example, we notice that a graph like Wiki-vote, shown in Figure 3.1(b), which has several social hubs [**?**], mixes faster than other graphs that do not exhibit such hubs clearly (e.g., Slashdot in Figure 3.1(c) and Epinion in Figure 3.1(a)). Also related to the structure, we notice that a random graph like Gnutella, shown in Figure 3.1(d), mixes faster than other graphs. This is initially not surprising, since random graphs are good expanders, which are also fast mixing [**?**]. The interesting observation, though, is that



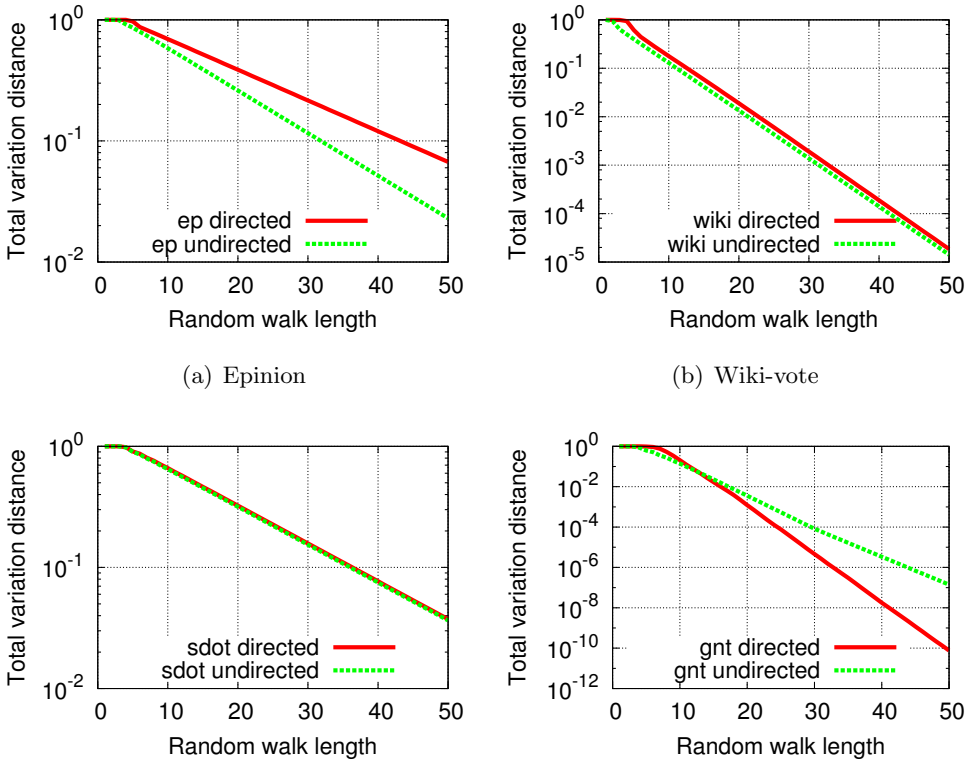

(a) Epinion

(b) Wiki-vote

(c) Slashdot

(d) Gnutella

Figure 3.1: The (mean) mixing time of directed graphs before and after omitting directions of edges. Each of the figures corresponds to the mean of measurements of $\epsilon$ that corresponds to the given random walk length for 1000 different initial distributions of sources in each social graphs.



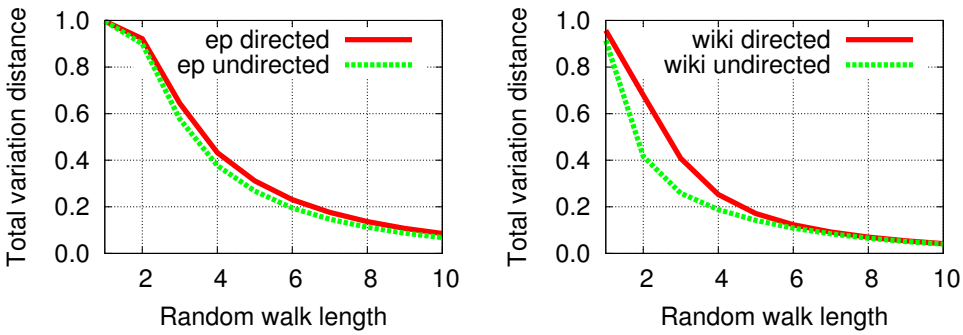

(a) Epinion

(b) Wiki-vote

(c) Slashdot

(d) Gnutella

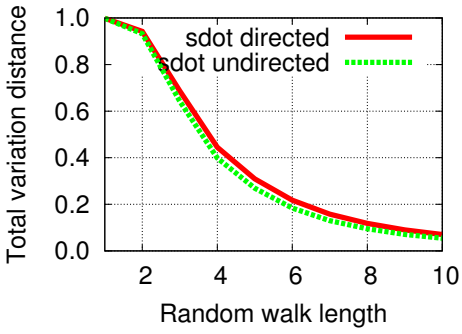
Figure 3.2: The (max) mixing time of directed graphs before and after omitting directions. Note that, by definition, these figures correspond to the mixing time of the social graph. While it bounds the mixing of all sources for which the mixing time is measured, it is less strictly representative of the quality needed for applications such Sybil defenses [**?**, **?**].



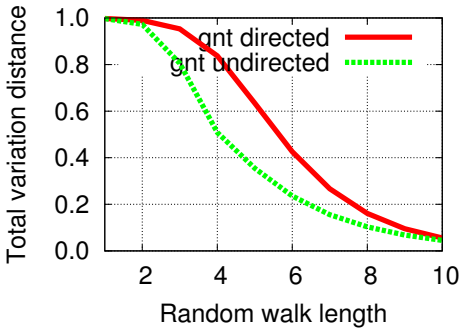

(a) Epinion




(b) Wiki-vote




(c) Slashdot




(d) Gnutella

Figure 3.3: The (mean) mixing time of directed graphs before and after omitting directions for short walks.

the same pattern of "fast-mixing" characteristics is preserved in Gnutella even when only considering a modified version of it; a strongly connected component. Furthermore, it is interesting to see how such expander "notion" applies to directed graphs (where the general notion and mixing time studies considered undirected expander graphs only).

The context of the graphs may affect their structure, and thus influence their mixing time. For example, we notice that two graphs with similar social contexts, like Slashdot and Epinion, have similar mixing times for longer walks (at average). This, however, is not shown in any of the other graphs. This similarity in the mixing pattern is exhibited clearer with short random walks, as shown in Figure 3.3.

In these measurements, and as we increase $w$, we find all possibilities of comparison exhibited—when comparing directed to undirected graphs: 1) no difference as in Wiki-vote graph in Figure 3.1(b), 2) undirected graphs mix better than directed graphs as in Slashdot and Epinion graphs in figures 3.1(c) and 3.1(a), and 3) directed graphs mix better than undirected graphs as in Gnutella in Figure 3.1(c).

While the two first cases are anticipated, the third case is surprising. It is unexpected to find that directed graphs mix better than undirected graphs in general. However, given that Gnutella has a special structure (originally a fixed degree graph), we hypothesize that omitting directions would alter the graph structure in unfavorable way to the mixing time. A potential case would be that adding more edges to the graph would divert long random walks on the graph. We further notice that this behavior is reverted when we restrict the random walk length to less than 10 steps.

Now we turn our attention to measuring the mixing time per the definition in Eq. ((3.2))—the results are shown in Figure 3.2. While the mean computed over all $\epsilon$'s for a given random walk length is meaningful for the average node—especially when evaluating systems built on top of social networks, it does not capture the worst case scenario which is of interest to the theoretical guarantees proposed in prior work in the literature that utilizes social graphs [**?**, **?**]. For example, for an anonymous communication system or asocial defense suggested on top of social networks, it is always better to prove guarantees with lower-bounds—which is actually used by Yu et al. in their Sybil defenses [**?**, **?**]. Lower bound guarantees are satisfied by the mixing time in the definition in Eq. ((3.2)), and computed as the maximum $\epsilon$ among all distributions for a given walk length. For the same experiment above, we plot the maximum $\epsilon$ for different

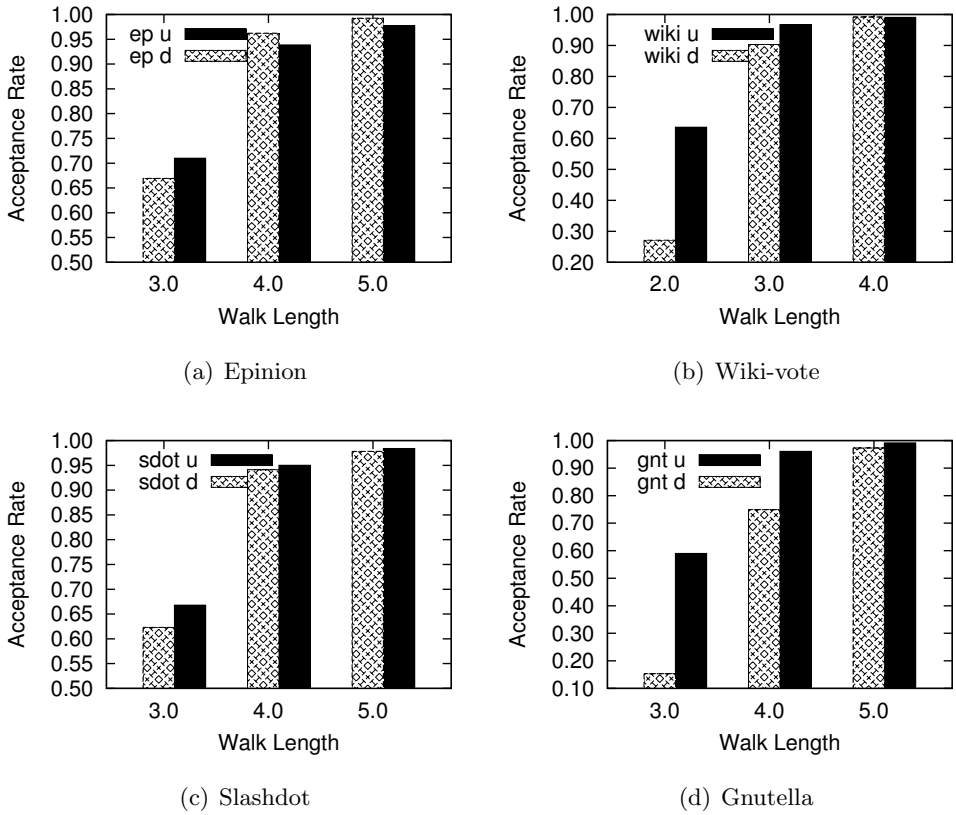
walk lengths in Figure 3.2.

While some of the patterns exhibited are consistent for the maximum $\epsilon$ in Figure 3.2 with the patterns in the average case in Figure 3.1, we observe that this pattern is switched in Slashdot and Wiki-vote. Particularly, we observe no difference in Slashdot, while the undirected Wiki-vote graph is faster mixing than the directed one, contrary to what is observed in the average case. Mohaisen et al. [**?**] argue that similar cases happen in other graphs, and may not be representative to the real mixing nature of these graphs. These cases happen when a set of nodes are sparsely connected to the majority of other nodes in the graph, thus slowing the mixing time of the whole graph [**?**].

We notice that Sybil defenses and anonymous communication systems built on social networks rely on shorter random walks than those shown in our experiments. Typically, a random walk used in a Sybil defense is in the order of $\log n$, where $n$ is the size of the graph, which translates into 10 to 15 for a graph of $100,000$ nodes [**?**]. We examine the mixing characteristics of these graphs in the needed range, for a random walk of proper length for the operation of these applications. We consider a walk length from 1 to 10 with 1 step increment and zoom-in the prior measurements discussed earlier for longer random walks (with the same settings as explained before). Interestingly, we find that *all* undirected graphs are faster mixing than directed graphs in that region of walk length, as shown in Figure 3.3. Surprisingly, we find that for a walk length of 5—the length we use for demonstrating impact of the mixing time on applications on social networks; see section 3.6—$\epsilon$ changes from 0.7 in the directed Gnutella to 0.3 in the undirected one. This difference, and in other graphs in Figure 3.3, has a great significance on social network-based applications.

## 3.6 Implications on Applications

In this section we explore the impact of the difference in the mixing of directed and undirected graphs on potential applications built on top of social networks that exploit the mixing time as their operation property. We consider SybilLimit [**?**] as a state-of-the-art representative work for Sybil defenses and "anonymity in the wild" [**?**] as an anonymous communication system.




(a) Epinion (b) Wiki-vote

(c) Slashdot (d) Gnutella

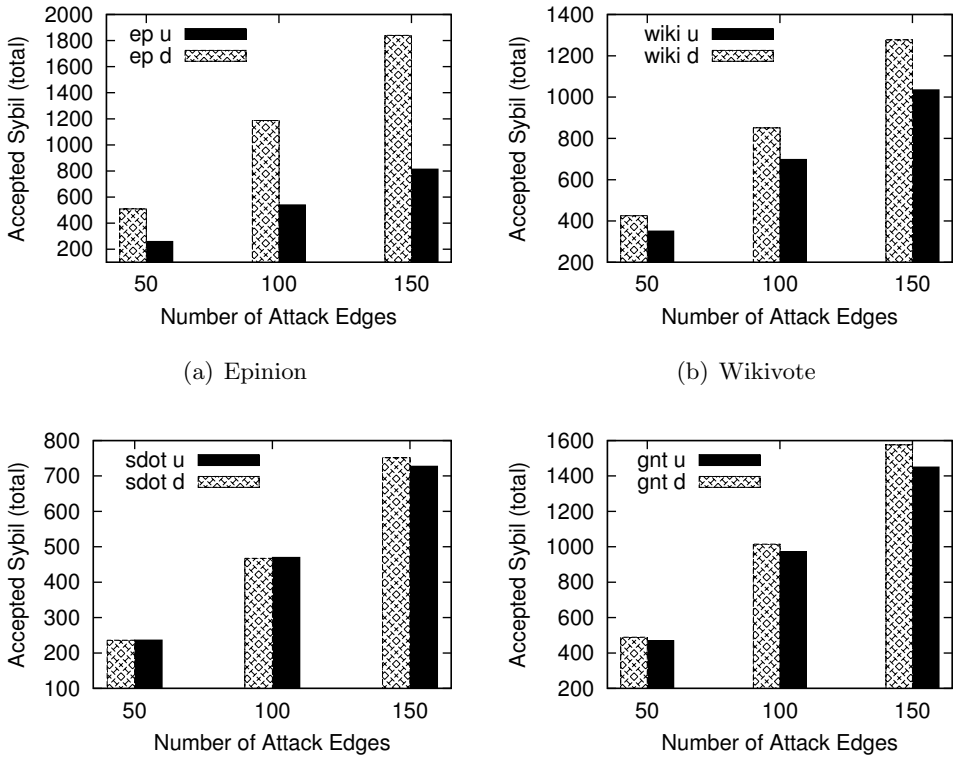
Figure 3.4: Acceptance rate of the honest nodes (suspects) by honest verifiers in directed and undirected graphs. Note that all undirected graphs outperform directed, except in Epinion.

### 3.6.1 Sybil Defenses

In the following we examine how the difference in the mixing time of directed and undirected social graphs impacts the performance of Sybil defenses operated on top of them. To demonstrate this impact, we choose SybilLimit [**?**].

#### 3.6.1.1 SybilLimit

In SybilLimit, each node samples $r$ edges in the graph as "witnesses", where $r = r_0\sqrt{m}$, by running $r$ independent instances of random walks each of length $w = O(\log n)$. Under certain assumptions on the graph's mixing characteristics, there is an overwhelming




(a) Epinion

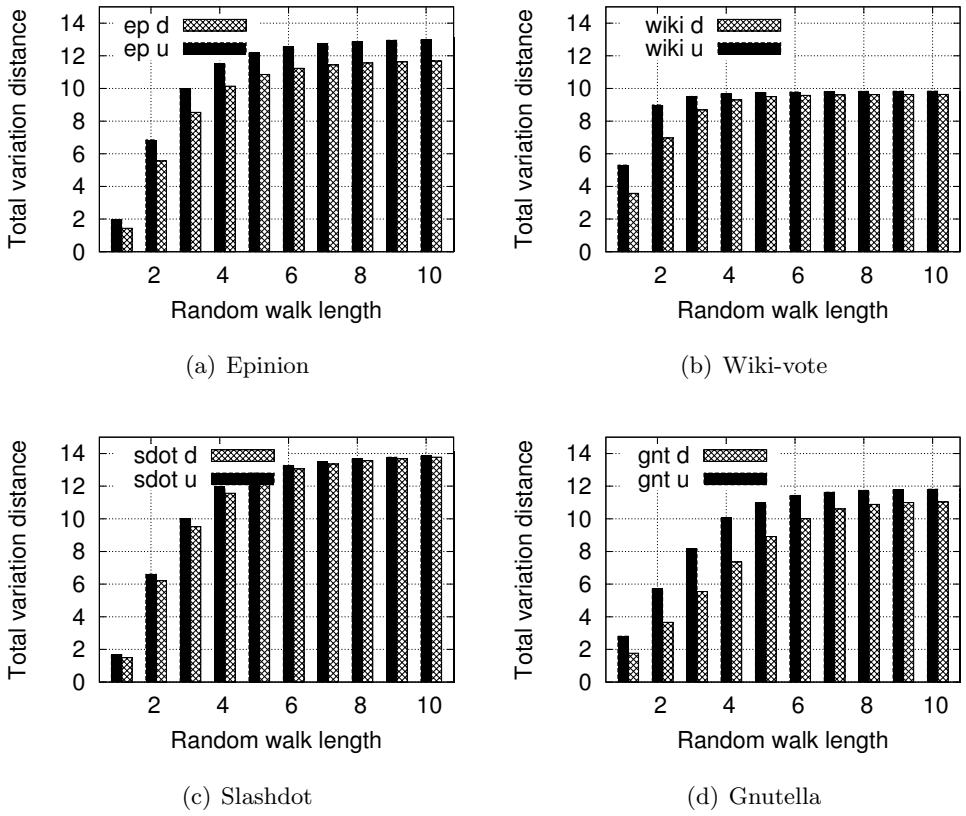
(b) Wikivote

(c) Slashdot

(d) Gnutella

Figure 3.5: Acceptance rate of the dishonest (Sybil) nodes by honest verifiers in directed and undirected graphs. Used walk lengths are 3, 4, 4, and 5 for Wiki-vote, Epinion, Slashdot, and Gnutella, respectively.

probability that the sampled subsets of honest nodes in the social graph will have a non-empty intersection, which would be used for suspect verification. Formally, if the social graph is fast mixing—i.e., has a mixing time of $O(\log n)$—then probability of the last node/edge visited in a walk of length $O(\log n)$ drawn from the edge/node stationary distribution is at least $1 - \frac{1}{n}$ (Theorem 1 in [**?**]). Accordingly, by setting $r_0$ properly, one can use the birthday paradox to make sure that the intersection between two sampled subsets of edges (by two honest nodes) is non-empty with an overwhelming probability. Furthermore, given that the social graph is fast mixing, and the number of attack edges—edges that connect Sybil with honest nodes—is limited, probability for random walks originated from honest region to dishonest region are limited. The impact of




(a) Epinion

(b) Wiki-vote

(c) Slashdot

(d) Gnutella

Figure 3.6: The mean entropy of random walks distributions on graphs before and after omitting directions for varying walk lengths ($H_w^u$ vs. $H_w^d$ for varying $w$ values).

such "escaping tails" on the operation of the defense is further marginalized using a "balance condition" which ensures that accepting a suspect would not cause a spike of the number of accepted suspects via a certain edge in the graph. Chances of dishonest nodes being accepted by sampling honest edges is limited, and bounded by the number of attack edges.

To evaluate the performance of SybilLimit, we use two evaluation metrics. We use acceptance rate of honest nodes, which is governed by the mixing characteristics of the social graph, and the acceptance of dishonest (Sybil) nodes, which is determined by both the mixing characteristic and the number of attack edges in the graph.

#### 3.6.1.2 Results and Discussion

In each directed and undirected graph, we use 1000 random nodes as suspect/verifier pairs (total of 499, 500 pairs of suspect/verifier in each graph). We then run SybilLimit and compute the average of both metrics of evaluation for the 1000 random verifiers. To further reduce randomness in the results and increase accuracy, we perform each experiment three times, by changing the 1000 random verifiers, and compute the average of the outcomes. In the following we summarize the main results and findings.

The results are shown in Figure 3.4 for the average honest nodes acceptance rate and in Figure 3.5 for the average of the total number of accepted Sybil nodes per the given number of attack edges, as the number of attack edges increases. In each of the figures, we make several observations. First, we observe that both metrics, the number of accepted honest and Sybil nodes are strongly correlated with the mixing characteristics of the different graphs, which is not surprising since the mixing time is the main property used for this application. By comparing Figure 3.3 to Figure 3.4 we observe that shorter random walks in faster mixing graphs are sufficient to accept all honest nodes. By comparing Figure 3.3 to Figure 3.5 and by fixing the length of attack edges (e.g., 100), we also observe that the number of Sybil identities introduced at average is strongly correlated with the mixing time; faster mixing graphs introduce more Sybil identities for the same number of attack edges.

We find that almost always directed graphs accept less honest nodes (by honest verifiers) as in Figure 3.4 and more Sybil identities (per attack edge) as in Figure 3.5. For example, we notice that for a random walk length 3 SybilLimit would accept about 95% of the honest nodes when operated on an undirected graph, where only 90% of these nodes are accepted when operated on the directed graph. Also, where operating SybilLimit on an undirected Gnutella graph with random walk length of 4 would allow accepting 95% of the honest nodes, it would only allow accepting 75% of the honest nodes on the same graph when considered directed.

An observation in favor of directed graphs is made on Epinion (in Figure 3.6(a)) when using walk length of 4. SybilLimit on a directed Epinion graph would accept roughly 95% honest nodes by honest verifiers (at average), whereas it would accept only 90% in the undirected graph case. We notice that, a lot more Sybils are introduced per attack edge for the same case (thrice the number of Sybils introduced per attack edge

in directed graphs).

We notice that introduced Sybils per attack edge are always more in directed graphs than in undirected graphs as shown in Figure 3.5. In some of these cases, the number of Sybils in directed graphs is three times more than in undirected ones—as in Figure 3.6(a), whereas other cases have almost no difference—as in Figure 3.6(a). Sizable, but not too large difference happens in the two other cases.

### 3.6.2 Anonymous Communication Systems

The idea of mixers over social links is very simple [**?**]. In these systems [**?**, **?**], users recruit their social acquaintance to relay their traffic and to provide anonymity to them. In the nutshell, each node (user) forwards her own traffic to her friends, and friends forward that traffic to their friends, and so on, for a certain number of hops, e.g. $w$. The number of hops $w$ is a system-wide parameter, which is determined by the security level desired in the system. The anonymity is defined for two parties; the sender and the receiver of traffic (we follow the same model in [**?**] for defining the anonymity of both parties).

#### 3.6.2.1 Anonymity Measures

For a sender, the anonymity defined in terms of the *anonymity set* is $n$, thus the entropy of the probability distribution of any node being the sender is $\log_2(n)$—same for both directed and undirected graphs. On the other hand, the anonymity set for a node being the receiver is determined by the probability distribution achieved after the fixed number of hops $w$ used in the system. Let the distribution of the final node selected in a *random walk* after $w$ hops be $\pi_i^w = \pi_i \mathbf{P}^w$, where $\pi_i^w = [\pi_i^w(j)]^{1\times n}$ ($\pi_i$ is an initial distribution). The anonymity of the receiver of the traffic (the last hop in the walk) is measured by the entropy $H_w$, which is given as

$$H_w = -\sum_{j=1}^{n} \pi_i^w(j) \log_2 \pi_i^w(j) \tag{3.6}$$

Using the entropy in Eq. ((3.6)), we define the *anonymity set* $A_w = 2^{H_w}$. The maximum entropy and anonymity set for a walk on a graph are achieved with the probability distribution of that walk as it approaches the stationary distribution.

We use $\overline{H_w^d}$ and $\overline{A_w^d}$ for the average entropy and anonymity sets in a directed graph, while $\overline{H_w^u}$ and $\overline{A_w^u}$ are used for the average entropy and anonymity sets in an undirected graph. We define the average entropy and anonymity sets for 1000 random walks starting from different sources (see below).

#### 3.6.2.2 Results and Discussion

Same as when we measured the mixing time and the performance of SybilLimit, we use 1000 initial distributions for each social graph. An initial distribution at node $v_i$ in a graph of $n$ nodes is a $1 \times n$ probability distribution of 1 at the $i$-th entry and 0 otherwise. We increase the length of the random walk $w$ from 1 to 10 with step increments of 1. At each time step we compute the probability distribution $\pi_w(i)$ of the random walk as $w$ increases, from which we compute the entropy as in above. We then compute the average entropy for the 1000 initial distributions (denoted in Table 3.4 as $\overline{H_w^d}$ and $\overline{H_w^u}$ for directed and undirected graphs, respectively). We plot the results of the entropy measurements in Figure 3.6. We observe a consistent pattern with measurements of the mixing time in section 3.5 and earlier results of SybilLimit in section 3.6.1. We find that walks on directed graphs have less entropy (at average) than in undirected graphs, for the same walk length. However, in some cases this difference is not big, indicating that the direction of edges in the graph may have insignificant impact.

If we set the random walk length to 6 in each graph, we obtain the results of average entropy and average anonymity sets shown in Table 3.4. We observe that while the difference in the entropy is small, one should keep in mind that the entropy is a sensitive measure, and a small difference in it would translate into large difference in the anonymity set. For example, the obtained pairs of entropy for directed and undirected cases for the same set of graphs, respectively, translate to the following pairs of anonymity sets: $(2411, 6013)$, $(764, 884)$, $(8551, 9891)$, and $(1026, 2759)$. The exponentially scaled measures of anonymity demonstrate the difference in both cases, which is related to the underlying graphs and their altered structure.

Table 3.4: The entropy and anonymity set comparison in directed and undirected graphs for random walk length $w = 6$ and for the different datasets.

| Dataset | $\overline{H_w^d}$ | $\overline{H_w^u}$ | $\overline{A_w^d}$ | $\overline{A_w^u}$ |
|---|---|---|---|---|
| Epinion | 11.236 | 12.554 | 2411 | 6013 |
| Wiki-Vote | 9.579 | 9.788 | 764 | 884 |
| Slashdot | 13.062 | 13.272 | 8551 | 9891 |
| Gnutella | 10.003 | 11.430 | 1026 | 2759 |

## 3.7 Summary

In this directon we have investigated mathematical tools and used them for measuring the mixing time of directed social graphs. Our work is mainly motivated by prior work in the literature on building social network-based applications and bringing insight on their operation by neglecting direction of edges in naturally directed social graphs. These applications are aimed to build services to improve security aspects of distributed systems, such as in Sybil defenses and anonymous communication. We show a consistent pattern in which directed graphs are in general slower mixing than undirected graphs for the specific parameters recommended by these applications. Furthermore, we show that these applications perform relatively poorly when the original property in the directed graphs is used.

While applications built on top of social networks may have a great potential in improving the security and reliability of distributed systems, we advocate that researchers have to pay more attention to the methods used for evaluating these applications. After all, it might not be possible, nor needed, to build a Sybil defense that accepts 95% or 99% of the honest nodes and to allow a small fraction of Sybils [**?**]. We observe that for a gain of 5% in its acceptance rate by undermining edge directions, a Sybil defense may hide a larger unseen cost: three folds the number of reported accepted Sybil identities per attack edge.

Our final recommendation is that, given most of these designs are intended for operation on undirected graphs, it is better to use undirected graphs—plenty of which are already available—for bringing insight on their operation rather than massaging

directed graphs. Simple alteration of graphs may greatly affect the underlying properties used for building such systems.

# Chapter 4

# Measuring Other Exploited Properties in Social Networks

Other basic properties used for building applications on top of social networks include the *betweenness centrality* of nodes in the social graph. The *betweenness centrality* [**?**] captures the significance of nodes to the flow between other nodes. Two types of betweenness centrality are known in literature: *shortest-path betweenness* [**?**, **?**, **?**] and *random walk-based betweenness* [**?**], although the shortest path betweenness is the one that is widely used. Another property used for building sybil defenses on top of social networks is graph expansion, as in Gatekeeper [**?**], where social graphs are assumed to have a well-expansion property.

In this chapter, we investigate by measurements the validity of both properties in social networks. We first show that the betweenness, the property used for building MobID [**?**] does not hold in social networks to support its operation. We then show that the expansion of social networks, and expected, is in line with the mixing properties of social networks. This is, fast mixing social graphs tend to have nice expansion properties whereas slower mixing graphs have less quality of expansion properties.

To complete the results in the prior two chapters, we explore the reason why certain social graphs are fast mixing, whereas others are slow mixing. We show that fast mixing graphs tend to have a single large core that is weakly affected by trimming nodes with smaller degrees from the graph (i.e., the graph is highly sparse). On the other hand,

slower mixing graphs dissolve and disappear after trimming the graph by removing nodes with small degree. Formally, we relate the mixing time by measurements to the $k$-coreness of the graph. We use these observations to suggest several heuristics to improve the mixing time of social graphs.

To this end, the organization of this chapter is follows. In section 4.1 we introduce preliminaries used in the rest of the chapter. Measurements are presented in chapter 4.2 (including how the quality of these properties affect Sybil defenses). In chapter 4.3 we show how $k$-coreness relate to the mixing time and suggest several heuristics to improve it. In section 4.4 we summarize findings in this chapter.

## 4.1 Preliminaries and Definitions

### 4.1.1 Betweenness

The ***shortest path betweenness*** weighs the significance of nodes as a result of their occurrence at the shortest path between other nodes by normalizing the number of paths between every two nodes in the network that pass through a particular node by the total number of such paths. While the shortest path betweenness requires a global knowledge of the whole topology—an issue that raises a lot of concerns in social network-based applications, the random walk based betweenness can be implemented at fully decentralized settings. Recall the random walk defined earlier and recall that the random walk connecting nodes $v_i$ and $v_j$, denoted as $v_i \overset{p_r}{\rightsquigarrow} v_j$, is the set of nodes where each node on the random walk is selected uniformly at random by its prior node.

We define the ***random walk-based betweenness*** for a node $v_i$ as the normalized expected number of times it is hit by the random walk. In short, the same model of the shortest path-based betweenness is used to compute the random walk-based betweenness by replacing the short path in the first one by the random walk in the latter one.The betweenness is used in [**?**] for defending against the Sybil attack and in [**?**, **?**] to improve routing in DTNs.

### 4.1.2 Similarity

The ***similarity*** is another property that is also used for building or improving the quality of applications that leverage social networks [**?**, **?**, **?**, **?**, **?**, **?**]. The similarity between two nodes in social networks is used for measuring the strength of social links and potentially predicting future interactions [**?**, **?**], or even weighting the value and significance of future interactions between nodes in social graphs [**?**, **?**, **?**]. Intuitive simple meaning of the similarity between two nodes in the context of a friend-to-friend social network is the number of friends common to both of them.A measure of similarity that reflects this intuitive meaning is the "cosine similarity" which computes how vectors are close to each other, each representing the row of a node in **A** [**?**].

### 4.1.3 Closeness

Another measure of centrality in graphs is the ***closeness***. The shortest path closeness captures how close is a node to others in the graph based on the expected length of the shortest pathes between that node and other nodes in the graph. Applications of such measure include routing on top of social networks [**?**, **?**].

### 4.1.4 Expansion

The (vertex) ***expansion*** of the graph is used in proving theoretical security guarantees of a Sybil defense. Let $S \subset V$, where $0 < |S| \leq 1/2|V|$, be any set of vertices in the graph. The (vertex) expansion factor $\alpha$ is defined as

$$\alpha = \min_{0<|S|\leq \frac{n}{2}} \frac{|N(S)|}{|S|} \tag{4.1}$$

Where the minimum is over all nonempty sets $S$ of at most $n/2$ vertices—$S$ need not be connected and thus the number of possible configurations of $S$ is exponential in $n$—and $N(S)$ is the set of vertices not in $S$ but each of which is connected by an edge to another node in $S$. In [**?**], where this property is used for building a Sybil defense, the definition of $S$ is further restricted so as $S$ is connected. Such restriction reduces the number configurations of $S$ to become linear in $n$. In the subsequent subsections, we elaborate on how to measure $\alpha$ in the latter context.

## 4.2 Measuring Other Properties

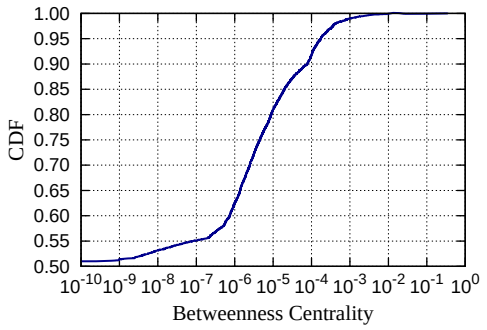
Here we outline our work on other properties used for trustworthy computing, the betweenness of nodes, and graph expansion.

### 4.2.1 Betweenness and Sybil Attack

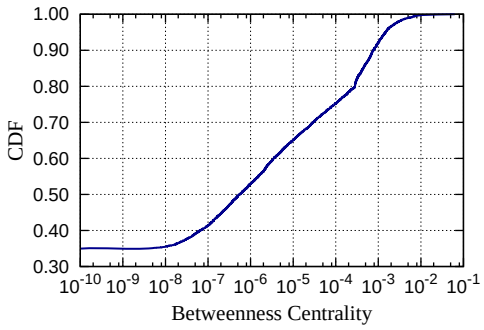
MobID [**?**], a social-network based Sybil defense has recently attracted attention because it newly provides a robust defense for mobile environments while existing defenses have largely been designed for peer-to-peer networks. Furthermore, MobID introduces *betweenness*, a graph-theoretic property in the social graph, as a metric of the goodness of nodes in order to defend against the Sybil attacks. By using this betweenness, MobID operates on two fundamental assumptions: i) highly enmeshed nodes in the social graphs have a nonzero betweenness, and ii) verifiers and suspects in an honest social graph have common friends. To understand the extent to which assumptions about the betweenness in the social networks is valid, we perform extensive experiments on several datasets. On each of these graphs, mostly shown in Table 6.2, we use the shortest path betweenness defined earlier. The betweenness of the nodes (as a CDF) for some of these graphs is shown in Figure 4.1 (Further details are in [**?**]). From these measurements, we observe that big proportion (35% to 50%) of the nodes in the social graph even without considering malicious nodes, have betweenness close to zero. Such finding is striking, in the sense that the betweenness alone cannot be used to build applications and make meaningful judgment on nodes in social networks to identify honest and malicious nodes.

### 4.2.2 Measurements of The Expansion

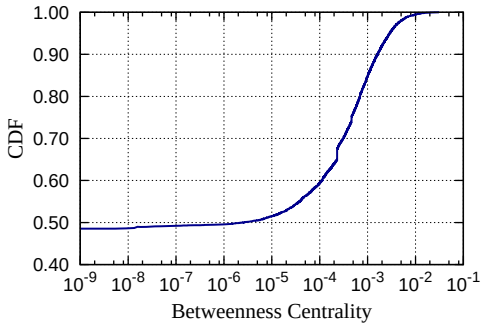
We estimate the expansion factor of a graph by constructing an envelope $Env_d$ (the same terminology used in [**?**]) formed by all nodes that are within a (shortest-path) distance $i$ from a core node. The expansion $Exp_i$ of the envelope consists of all of its immediate neighbors. We define the expansion factor as $\alpha_i = |Exp_i|/|Env_i|$. In our experiments, by letting each of the nodes in the graph to be the core, we calculate the expansion factor $\alpha_i$, with $0 \leq i \leq d-1$, where $d$ is the diameter of the graph, by building a tree rooted at the core that expands in the breadth-firth search manner. Let $L_i$ be




(a) Youtube




(b) Wiki-vote




(c) Physics 3

Figure 4.1: Preliminary measurements of the betweenness of nodes in different social graphs.

the number of nodes at level $i$ in the tree, we have

$$\alpha_i = \frac{L_{i+1}}{\sum_{j=0}^{i} L_j} \tag{4.2}$$

To estimate the expansion characteristics of the social graphs, we run our experiment by letting each node in the graph to be the core and build a breadth-first search tree rooted at that core. We then count the number of nodes in each level of the tree to calculate the expansion factor as in Eq. (4.2).

To motivate for this measurement, consider the measurements in Table 4.1. In this experiment, we run Gatekeeper [**?**] on four different datasets with different characteristics. Unsurprisingly, we observe that results of operating Gatekeeper on such graphs are

Table 4.1: Numerical results of operating Gatekeeper [**?**] on top of different social graphs with different characteristics. 10 Attackers are selected randomly and 99 distributers are sampled in each case (attack edges are $131, 145, 277$, and 344, respectively). $f$ is a security parameter, honest acceptance percent is of the whole graph size and Sybil is per attach edge.

| Dataset | Accept. | $f = 0.1$ | $f = 0.3$ | $f = 0.5$ |
|---|---|---|---|---|
| Physics 1 | Honest | 89.90% | 70.50% | 54.40% |
| | Sybil | 8.4 | 1.7 | 0.7 |
| Facebook | Honest | 98.40% | 79.00% | 51.30% |
| | Sybil | 10.1 | 1.8 | 0.7 |
| LiveJournal | Honest | 97.00% | 78.60% | 53.20% |
| | Sybil | 3.7 | 0.7 | 0.3 |
| Slashdot | Honest | 97.00% | 81.10% | 55.20% |
| | Sybil | 3.1 | 0.8 | 0.4 |

quite anticipated given that that such graphs are experimented on other social network-based Sybil defenses (in [**?**] and [**?**]), despite that other defenses require social graphs to be fast-mixing whereas GateKeeper requires the graphs to be expanders with good expansion factor. Hence, we establish that the property in action is closely related to the mixing time.

Following the model in (4.2), we construct a breadth first search tree for each source and compute its expansion as we go down the stream in the search tree. For all sources in the graph, where each node is considered as a source of expansion, the running time of a naive implementation of our algorithm takes $O(nm)$, which is a manageable overhead for small to medium sized social graphs.

We develop this algorithm to compute the expansion for all datasets in Table 6.2. Each source node has $d - 1$ measurements of sets of nodes and their neighbors, where $d$ is the diameter of the graph. To visualize the tendency of the expansion as the set size increases, we aggregate the unique sizes of sets, and find the number of neighbors to each of them. For each set of neighbors to a unique size of nodes, we compute the maximum, minimum, and expected number of neighbors. These statistics of measurements for a

selected set of datasets are shown in Fig. 4.2.

To further investigate the expansion of social graphs when compared to each other, we use the model in (4.2). For all sets of nodes with the same size, we compute the expected expansion as the average of all sizes of different neighbor sets. The result of the average expansion is shown is shown in Figure 4.3. The reader should keep in mind that these curves and $\alpha$ for each graph needs to be consider in relation with the graph size. For example, where one would see that slashdot's curve is above the curve of Wiki-vote, suggesting that expansion characteristics of Slashdot are better than wiki-vote, the size of wiki-vote is one-tenth the size of Slashdot.

## 4.3 $k$-Coreness and the Mixing Time

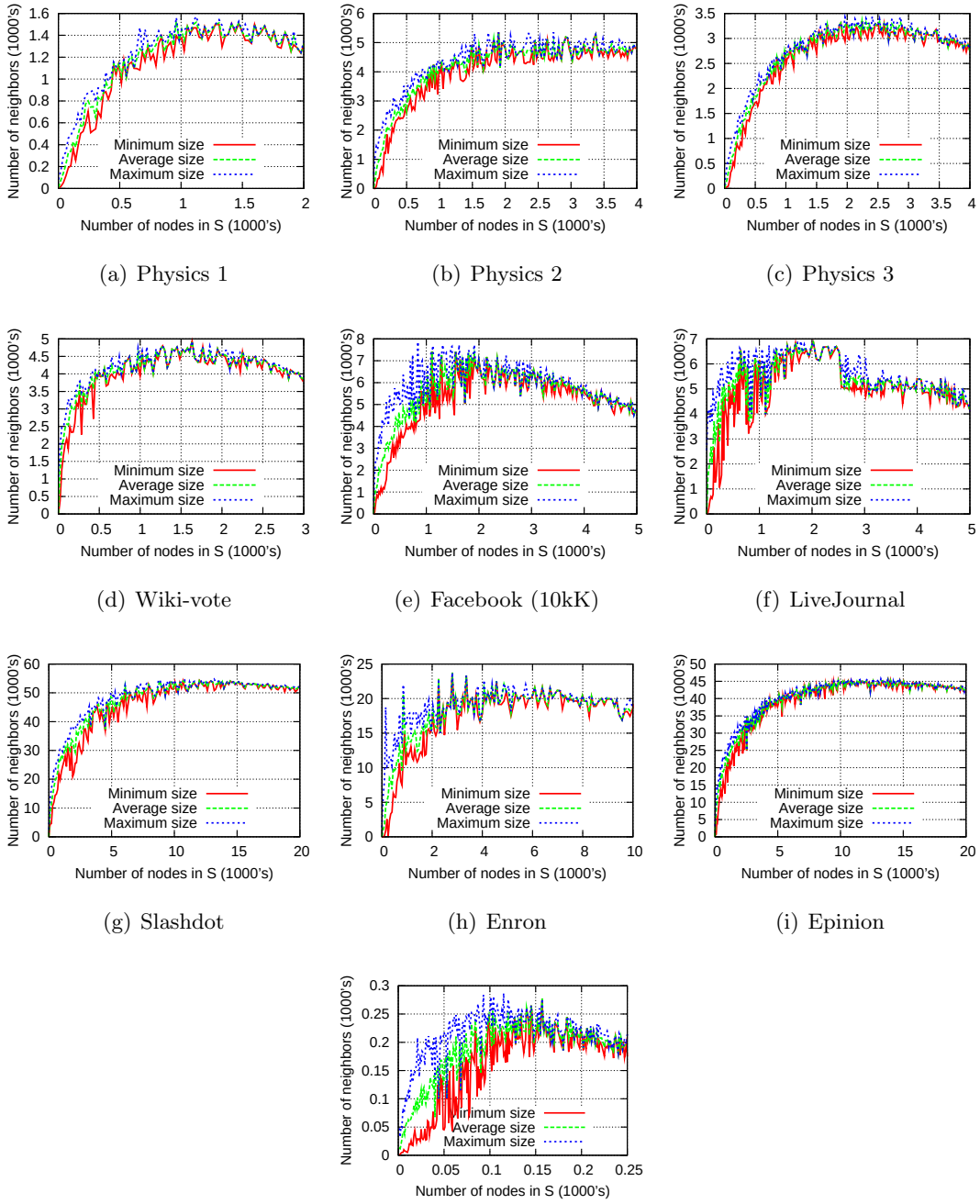
A natural question following our measurements of the mixing time would be: what make a fast mixing time graph, fast mixing. Indeed graph measures, like conductance and modularity are very related to the mixing time. Here we seek a simple method to compute and indicate the mixing characteristics of these graphs [1] .

### 4.3.1 The Mixing Time and $k$-Coreness

Among the measures of graph cohesiveness is $k$-coreness measure. For an undirected graph $G = (V, E)$ as defined earlier, and for any $k$, let $G_k = (V_k, E_k)$ be a subgraph in $G$ such that $|V_k| = n_k$, and with the constraint that for all $v_i \in V$, the minimum degree of any node $v_j \in V_k$ is $k$. $G_k$ is said to be a $k-$core of $G$ if, in addition to the above condition, it is a maximal and connected graph. By relaxing the connectivity condition, we get a set of cores (potentially more than one) each of which satisfies the degree condition. For such (potentially disconnected) $k$-core, we define the normalized size as $n_k/n$.

An efficient algorithm for decomposing a simple graph to its $k-$cores by iteratively pruning nodes with degree less than $k$ has the complexity of $O(m)$ [**?**]. The definition of $k$-core [**?**] is equivalent to $k$-coloring [**?**]. For more details, see [**?**].

[1] We limit attention in this section to understanding the mixing time in undirected social networks. The results can be extended to directed graphs, benefiting from community definitions in these graphs as in [**?**].



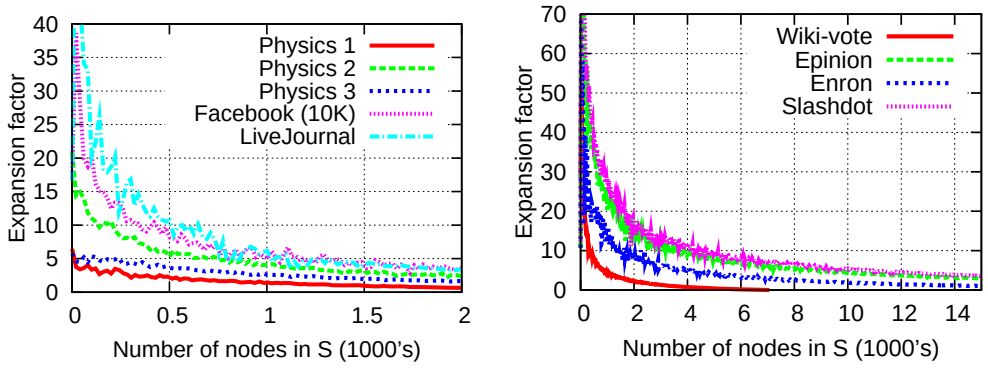

(a) Physics 1 (b) Physics 2 (c) Physics 3

(d) Wiki-vote (e) Facebook (10kK) (f) LiveJournal

(g) Slashdot (h) Enron (i) Epinion

(j) Rice-Grad

Figure 4.2: A measured of the expansion of sets of nodes of different sizes beginning from every nodes in the given graphs as potential core for the expansion.




(a) Small datasets

(b) Medium datasets

Figure 4.3: Expected expansion for various set sizes of various social graphs.

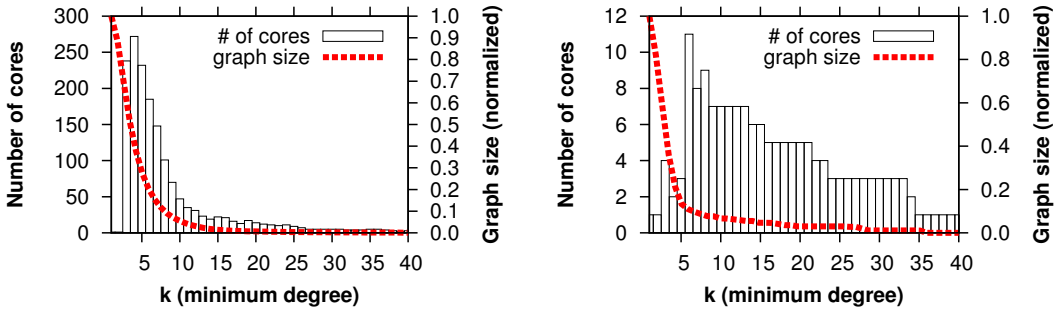
### 4.3.2 Measurements and Results

We use some of the datasets in Table 6.2 to demonstrate the relationship between core structure and the mixing time of social graphs. In short, the mixing characteristics of these graphs are shown in Figure 2.7, for the average case of the mixing time.

For each of these graphs we use an off-the-shelf implementation of the linear-time algorithm in [**?**] to compute the $k-$core by relaxing the connectivity assumption as described above. As $k$ increases to its ultimate value at which the graph disappears, we compute the following: (1) the number of cores in each $k$-core, (2) the normalized size of each $k$-core. The results of these measurements are shown in Figure 4.4 and Figure 4.5 [**?**]. Notice that graphs in Figure 4.4 are slow mixing and graphs in Figure 4.5 are fast mixing, as demonstrated in Figure 2.7 for average mixing.

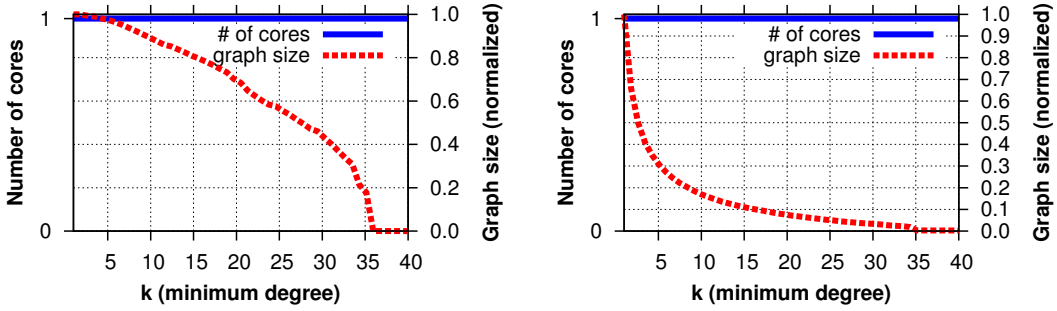
By comparing Figures 4.4 and Figure 4.5, we observe that slow mixing graphs are less cohesive whereas fast mixing graphs are more cohesive. This is reflected on the number of cores in the $k$-core of each graph as we increase $k$ till the graph is dissolved entirely. Also, whereas slow mixing graphs—shown in Figure 4.4—are decomposed into multiple cores as we increase $k$, fast mixing graphs resist this decomposition and remain cohesive as we increase $k$.

Second, despite that slow mixing graphs are decomposed into multiple cores, these cores are relatively small in size and the disappearance of the graph is very quick in many cases as we increase $k$. Fast mixing graphs have on the other hand remain in a




(a) DBLP (b) Physics 1

Figure 4.4: Core structure (slow mixing social graphs).




(a) Wiki-vote (b) Slashdot

Figure 4.5: Core structure (fast mixing social graphs).

single core, which is relatively larger in size than the counterpart core in slow mixing graphs—see [**?**] for more findings on core structure and experiments with other social graphs.

### 4.3.3 Improving the Mixing Time

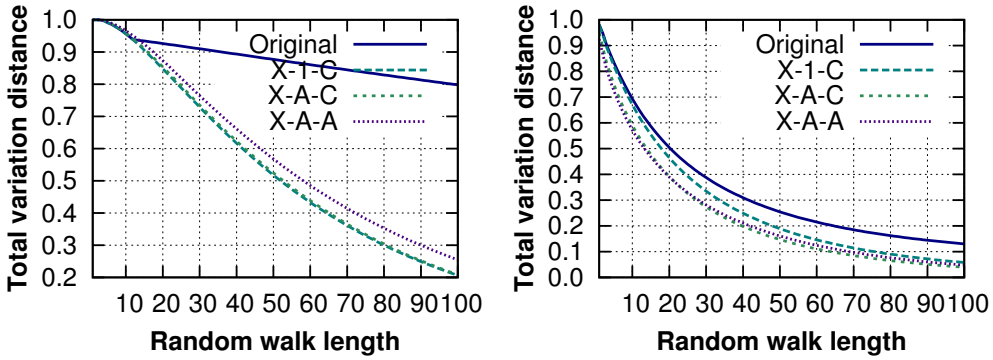
Here, we propose several heuristics to improve the mixing time of slow-mixing social graphs using their structural properties which are discussed earlier in section 4.3.2. Each of the following heuristics aims to prevent the creation of multiple cores as $k$ increases using auxiliary edges. We call the process "core wiring". We introduce these heuristics with Sybil defenses [**?**] in mind as potential applications. We refer to the largest core as the *major core* and any other core is a *minor core*.

[1] **Heuristic X-1-C:** wires a single node in each minor core $X$ with a node in the major core $C$ using *one* edge. The end vertices of added edges can be arbitrarily chosen. By doing so, we can easily see that the graph will always have a single core at any time while increasing $k$. The number of added edges is the sum of the number of cores in each $k$-core, for all $k$, minus $k$.

[2] **Heuristic X-A-C:** wires each node in each of the minor cores with a node in the major component, as we increase $k$. Same as above, this would prevent producing multiple cores at time and the number of auxiliary edges is bounded by the *number of nodes* in the minor components.

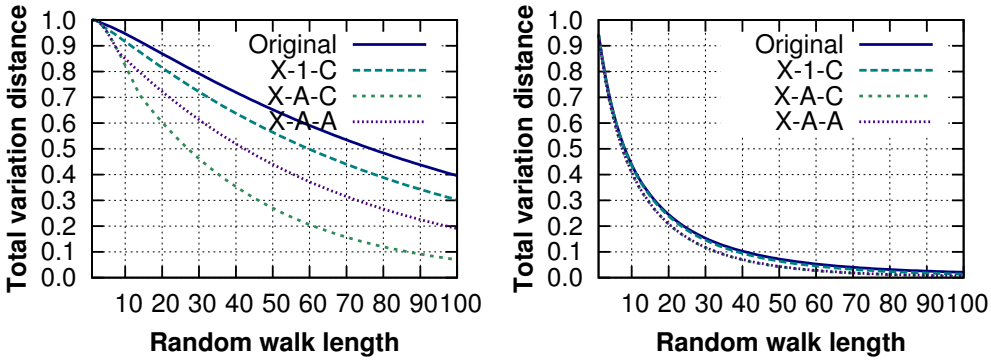
[3] **Heuristic X-A-A:** wires all nodes in a minor core to other cores in the graph, including both minor and major cores. The number of auxiliary edges is bounded by the *order* of the number of nodes in each $k$-core.

For further discussions on the rational of this method, in relation with prior literature work, see [**?**]. In short, auxiliary edges added in our heuristics can be made part of the evolution of the social graph through link recommendation. Alternatively, when centralized systems are built on top of social networks, these edges can be virtually created among honest nodes if labels of nodes are given.




(a) Max (b) Mean

Figure 4.6: Mixing time measurement of Physics 1 before/after improving its mixing characteristics.




(a) Max (b) Mean

Figure 4.7: Mixing time measurement of Physics 2 before/after improving its mixing characteristics.

## 4.4 Summary

Other less popularly used properties for building systems on top of social networks are the expansion and betweenness, although as the mixing time they are never measured at scale to verify their usability in this context. We measure both properties and show that whereas the betweenness in social networks does not hold with the quality required for building such applications, the expansion hold in the same way that the mixing time holds in it; i.e., slower mixing graphs have poor expansion properties whereas fast mixing graphs have good expansion.

We complete the results in the prior two chapters and investigate heuristics to improve the mixing time of social networks by auxiliary links. These links are located in the social networks at critical sections that dissolve by trimming lower degree nodes. We show, by measurements, that adding these links (or edges) would prevent these graphs from dissolving and improve the mixing time.

# Chapter 5

# Incorporating Differential Trust into Social Network-based Sybil Defenses

In most of the literature that considered social networks for building Sybil defenses, as well as other applications that rely on social networks' trust and structure, the simple uniform random walk highlighted earlier in the context of measuring the mixing time is used. As we have observed earlier, the mixing time of social graphs depends greatly on the underlying social graphs and there is a negative association between the strength of the social links and the mixing time. In part of the project, we propose to investigate several designs of modulated random walks that consider a "trust" parameter between nodes in order to tune the performance of the designs built on top social network, which use the random walk theory. In all of the proposed random walks, the purpose is to assign "trust-driven" weights and thus deviate from uniform random walk. Supported by our work and results in [**?**], we do this by either capturing the random walk in the originator or current node, as the case of originator-biased and lazy random walks, or by biasing the random walk probability at each node, as the case of interaction and similarity-biased random walks, or a combination of them. The intuition of the lazy and originator-biased random walk is that nodes trust "their own selves" and other nodes within their community more than others. On the other hand, interaction and

similarity-biased trust assignments try to weigh the natural social aspect of trust levels.

It is worth noting that while this direction is motivated by the need for incorporating trust in social network-based Sybil defenses, the designs below are not limited to these Sybil defenses but can also be used without any modifications for any random walk based algorithm on graphs, including random walk based community discovery and random walk-based betweenness [**?**, **?**]. Furthermore, these designs can be naturally and easily extended to other applications that use social networks as bootstrapping graphs, with minimal modifications. For example, the trust or distrust of nodes among each other can be incorporated as a random process driven from these models to identify decisions on collaboration among nodes. This collaboration can be used to understand the implication on the behavior of the different designs built on top of social networks. This latter direction is partly what we want to investigate further as part of the proposed work in project.

## 5.1 Designs to Account for Trust

Given the motivation for these designs, we now briefly describe them by deriving $\mathbf{P}$ and $\pi$ required for characterizing them. We omit the details for lack of space (see [**?**] for the complete proofs, further experiments, and discussions).

### 5.1.1 Lazy Random Walks

To accommodate for the trust exhibited in the social graph, for simplicity we assume a global single parameter $\alpha$ in the network which is used to characterize this trust level (shown in Figure 5.1). We use this parameter in the different schemes to enforce and apply the trust along with other parameters used for ensuring the (e.g., driven from the algorithmic property in the graph). The transition matrix

$$\mathbf{P'} = \alpha\mathbf{I} + (1-\alpha)\mathbf{P} \tag{5.1}$$

which yields a transition according to $p_{ij}$ defined as follows:

$$p_{ij} = \begin{cases} \frac{1-\alpha}{\deg(v_i)} & v_j \in N(v_i) \\ \alpha & v_j = v_i \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

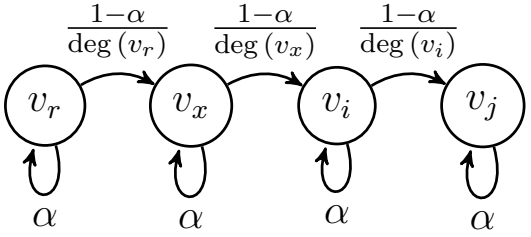



Figure 5.1: An illustration of the lazy random walk. For simplicity, $\alpha$ is equal for each node though it can be determined by each node locally to reflect what that node perceive as the trust of the network.

We note that for the transition probability defined in ((5.1)), by adding self loops it does not alter the final stationary distribution from that in the uniform random walk. Further details on the proof of this are in [**?**].

### 5.1.2 Originator-biased Random Walk

We incorporate the concept of biased random on the social graph walks to characterize the bias introduced by the trust among different social actors (nodes). At each time step, each node decides to direct the random walk back towards the node that initiates the random walk, i.e., node $v_r$, with a fixed probability $\alpha$ or follow the original simple random walk by *uniformly* selecting among its neighbors with the total remaining probability $1-\alpha$ (a diagram of the walk is shown in Figure 5.2). The transition probability that captures the movement of the random walk, initiated by a random node $v_r$, and moving from node $v_i$ to node $v_j$ is defined according to $p_{ij}$ as follows

$$p_{ij} = \begin{cases} \alpha & j = r, v_r \notin N(v_i) \\ \alpha + \frac{1-\alpha}{\deg(v_i)} & j = r, v_r \in N(v_i) \\ \frac{1-\alpha}{\deg(v_i)} & j \neq r, v_j \in N(v_i) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

We note that, unlike the lazy random walks, the transition probability here considers moving the state back to the originator of the random walk, a state that may not be connected to the current state in the social graph. This requires a virtual connection between each node through the walk – every node in the graph – and each originator

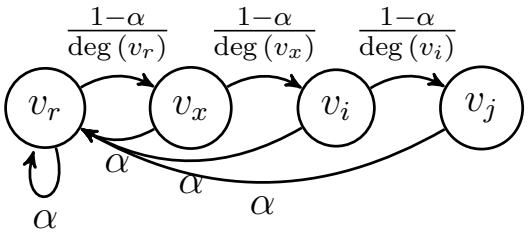



Figure 5.2: An illustration of the originator biased random walk.

of a random walk. To mathematically model this transition loop, for each node $v_r (1 \leq r \leq n)$, we define $\mathbf{A}_r$ as an all-zero matrix with the exception of the $r^{\text{th}}$ row which is 1's. Using $\mathbf{A}_r$, we further define the originator-biased transition matrix, for the walk originated from $v_r$, as

$$\mathbf{P'} = \alpha \mathbf{A}_r + (1 - \alpha)\mathbf{P}. \tag{5.4}$$

We can show that $\mathbf{P'}$ is stochastic since each row in it sums to 1. Furthermore, since $\mathbf{P'}$ depends on the initial state $v_r$, we observe that the "stationary" distribution is not unique among all initial states, and so we refer to it as the "bounding distribution" for the walk initiated from $v_r$. The bounding distribution in that case is $\pi^{(v_r)} = [\pi_i]^{1 \times n}$ where $\pi_i$ is

$$\pi_i = \begin{cases} (1-\alpha)\frac{\deg(v_i)}{2m} & v_i \in V \setminus \{v_r\} \\ \alpha + \frac{\deg(v_i)}{2m} & v_i = v_r \end{cases} \tag{5.5}$$

We note also that the bounding distribution in ((5.5)) is a valid probability distribution since it satisfies the distribution conditions (sums to 1 and invariant to $\mathbf{P'}$). Details on the proof are in [**?**].

### 5.1.3 Interaction-biased Random Walk

The interaction between nodes can be used to measure the strength of the social links between nodes in the social network [**?**]. In this model, high weights are assigned to edges between nodes with high interaction and low weights are assigned to edges between nodes with low interaction. Formally, let $\mathbf{B}$ be the raw interaction measurements between

nodes in $G$ and $\mathbf{D}$ be a diagonal matrix representing the row norm of $\mathbf{B}$. The transition matrix $\mathbf{P}$ of the random walk based on interaction is then computed as $\mathbf{P'} = \mathbf{D}^{-1}\mathbf{B}$. The stationary distribution of the random walk on $G$ following to the probability in $\mathbf{P'}$ is $\pi = [\pi_i]^{1\times n}$ where

$$\pi_i = (\sum_{j=1}^{n}\sum_{k=1}^{n} b_{jk})^{-1}(\sum_{z=1}^{n} b_{zi}). \tag{5.6}$$

We observe that this distribution makes a valid probability distribution since $\sum_{i=1}^{n} \pi_i = 1$ and is a stationary distribution since $\pi\mathbf{P} = \pi$.

Wilson et al. [**?**] introduced a slightly different model to capture interaction between nodes in the social graph. The interaction graph $G' = (V, E')$ is defined for a social graph $G = (V, E)$ where $E' \subseteq E$ and $e_{ij} \in E'$ if $I(v_i, v_j) \geq \delta$, where $I$ is an interaction measure to assign weights on edges between $v_i$ and $v_j$ for all $i, j$, and $\delta$ is a threshold parameter. The interaction measure used in [**?**] is the number of interactions over a period of time. This later model further simplifies the random walk where the $\mathbf{P'}$ is defined over $G'$, as well as the stationary distribution. In our measurements, we use this model and some of the datasets used in Wilson et al.'s work, though we do not exclude to further investigate the characteristics and potential of non-threshold based interaction model (the one described above) in the near future.

### 5.1.4 Similarity-biased Random Walk

The similarity between social nodes in social networks is used for measuring the strength of social links and predicting future interactions [**?**, **?**]. For two nodes $v_i$ and $v_j$ with sets of neighbors $N(v_i)$ and $N(v_j)$, respectively, the similarity is $\frac{N(v_i)\cap N(v_j)}{N(v_i)\cup N(v_j)}$. For $\mathbf{a}_i$ and $\mathbf{a}_j$, two rows in $\mathbf{A}$ corresponding to the entries of $v_i$ and $v_j$, we use the cosine similarity measure given as $S(v_i, v_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{|\mathbf{v}_i|_2 |\mathbf{v}_j|_2}$, where $|\cdot|_2$ is the L2-Norm. To avoid disconnected graphs resulting from edge cases, we augment the similarity by adding 1 to the denominator to account for the edge between the nodes. Also, we compute the similarity for adjacent nodes only, so that $\mathbf{S} = [s_{ij}]$ where $s_{ij} = S(v_i, v_j)$ if $v_j \in N(v_i)$ or 0 otherwise. The transition matrix $\mathbf{P}$ of a random walk defined using the similarity is given as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{S}$ where $\mathbf{D}$ is a diagonal matrix with diagonal elements being the row norm of $\mathbf{S}$. Accordingly, the stationary distribution of random walks on $G$ according to

$\mathbf{P}$ is $\pi = [\pi_i]^{1\times n}$ where $\pi_i = (\sum_{z=1}^{n} s_{zi})(\sum_{j=1}^{n}\sum_{k=1}^{n} s_{jk})^{-1}$.

## 5.2 Implication of the Designs on the Mixing Time

Along with the simple random walk-based design, we implement three of the proposed designs: lazy, originator, and similarity biased random walks. We use the simple random walk-based implementation over the interaction graph of Wilson et al.'s [**?**] to learn the performance of the interaction-based model. We examine the impact of each design on the mixing time on some graphs from Table 5.1. The results are shown in Fig. 5.3 and Fig. 5.4. We observe that, while they bound the mixing time of the different social graphs, the originator-biased random walk is too sensitive even to a small $\alpha$. For example, as in Fig. 5.4(a) for Facebook social graph in Table 5.1, $\epsilon \approx 1/4$ is realizable at $w = 6$ with the simple random walk, $w > 10$ for both lazy and originator-biased random walk. However, this happens with $\alpha = 0.5$ in the lazy against $\alpha \approx 0.1$ in the originator-biased walk. This observation is made clearer on Fig. 5.4 which compares the mixing time of four different social graphs with different characteristics when using the simple and modified random walks.

We also observe in Fig. 5.3 and Fig. 5.4 that the linear increments in the parameters do not necessarily have linear effect on the measured mixing time. Furthermore, this behavior is made clearer in the experiments performed on SybilLimit and shown in Fig. 5.5 and Fig. 5.6. This however is not surprising, at least with the originator-biased random walk since the probability of intersection when sampling from the stationary distribution is $\leq e^{-8(1-\alpha)^4}$ from which one can see the exponential effect of $\alpha$ on the admission rate. While this explains the general tendency in the admission rates of SybilLimit, it does not answer some inconsistency shown in Fig. 5.6(b) for the transition between $\alpha = 0.12, 0.16$, and $0.20$. One additional explanation for that is the community structure in this graph, which is shown in [**?**] to be clear in Physics 1 and problematic for Sybil defenses (results for the same graph are in Fig. 5.5(b) and Fig. 5.6(b)). On the other hand, some graphs are less sensitive to the same value of these parameters, e.g., Facebook with the results shown in figures 5.3(a), 5.4(a), 5.5(d), and 5.6(d). One possible explanation for this behavior is that this graph has less community structure. Reasoning about this behavior and its quantification is to be our future work.

Table 5.1: Social graphs with their size, diameter, and radius. Physics 1, 2, 3 are relativity, high energy and high energy theory co-authorship respectively.

| **Social network** | **Nodes** | **Edges** | **Diameter** | **Radius** |
|---|---|---|---|---|
| Physics 1 [?] | 4,158 | 13,428 | 17 | 9 |
| Sdot [?] | 10,000 | 14,6469 | 6 | 3 |
| Physics 2 [?] | 11,204 | 117,649 | 13 | 7 |
| Physics 3 [?] | 8,638 | 24,827 | 18 | 10 |
| Wiki-vote [?] | 7,066 | 100,736 | 7 | 4 |
| Enron [?] | 10,000 | 108,373 | 4 | 2 |
| Epinion [?] | 10,000 | 210,173 | 4 | 2 |
| DBLP [?] | 10,000 | 20,684 | 8 | 4 |
| Facebook [?] | 10,000 | 81,460 | 4 | 2 |
| Livejournal [?] | 10,000 | 135,633 | 6 | 3 |
| Youtube [?] | 10,000 | 58,362 | 4 | 2 |
| Rice-cs-grad [?] | 501 | 3255 | 9 | 5 |
| Rice-cs-ugrad [?] | 1221 | 43153 | 6 | 3 |

### 5.2.1 Sybil Defense Performance Over Simple Random Walks

To understand the necessary mixing time quality required for the operation of SybilLimit, we measure the performance of SybilLimit using simple random walks, where the evaluation metric is the percent of honest nodes accepted by other honest nodes. For each walk with length $w(0 \leq w \leq 30)$, we compute the number of accepted nodes as a percent out of $n(n-1)$—total verifier/suspect pairs. Since SybilLimit accepts nodes on edges only, it works for $w \geq 2$. The results are shown in Fig. 5.7 and the variable mixing time shown earlier is further highlighted by observing the percent of accepted nodes when varying $w$. We observe that, unlike claims in SybilLimit where one would expect 95% admission rate at $w = 4$, some graphs require $w = 30$; where graphs which admit high percent of nodes for small $w$ are those with poor trust.

### 5.2.2 Defense Performance with Modified Walks

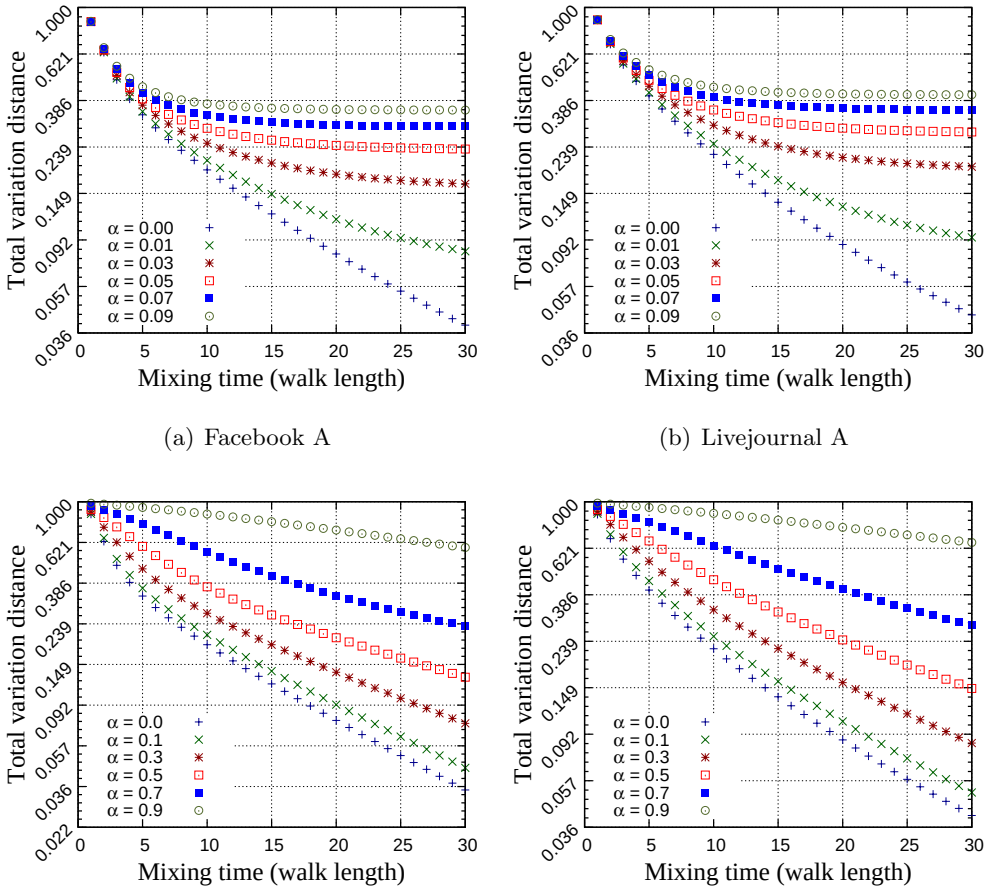
Now we study the impact of the modified random walks on the performance of SybilLimit. We select four datasets with different characteristics from Table 5.1: DBLP, Facebook, Facebook (Rice grad), and Physics 1 (relativity theory). We implement modified SybilLimit versions that consider changes introduced by the modified random walks and test the admission rate of honest nodes under different values of $\alpha$ and $w$.

#### 5.2.2.1 Performance Over Lazy Random Walk

we measure the performance of SybilLimit operating with the lazy random walks – results are shown in Fig. 5.5. We vary $w$ from 0 to 30 with steps of 2. We further vary $\alpha$ associated with the lazy random walk from 0 to 0.80 with steps of 0.16—$\alpha = 0$ means simple random walk. While the performance of SybilLimit is generally degraded when increasing $\alpha$, we observe that the amount of degradation varies and depends on the initial quality of the graph. For example, by comparing DBLP (Fig. 5.5(c)) to Facebook (Fig. 5.5(d)) we observe that for $w = 10$, DBLP and Facebook admit about 97% and 100% of the honest nodes respectively for $\alpha = 0$. For the same $w$ and $\alpha = 0.64$, the accepted nodes in Facebook are still close to 100% while the accepted nodes in DBLP are only 50% suggesting variable sensitivity of different graphs to same $\alpha$. Once we raise $\alpha$ to 0.80, the number of accepted nodes in Facebook decreases to 80% while giving only 25% in DBLP. One explanation of this behavior is what we have discussed in section 5.2. Also, since the ultimiate goal of this model is to characterize trust, which already differs in these graphs, we know that $\alpha$ should not necessarily be equal in both cases. For instance, if one is concerned about achieving same admission rate for the same $w$ in both cases, one may choose $\alpha = 0.48$ in DBLP and $\alpha = 0.80$ in Facebook where $w = 10$ in both cases which yields 80% admission rate in both cases.

#### 5.2.2.2 Performance Over Originator-biased Random Walk

The same settings in section 5.2.2.1 are used in this experiment but here we vary $\alpha$ from 0 to 0.2 with 0.02 steps since the originator-biased walk is more sensitive to smaller $\alpha$ than the lazy-random walk. Similar to the lazy walk, the originator-biased walk, as shown in Fig. 5.6, influences the performance of SybilLimit on different graphs differently,



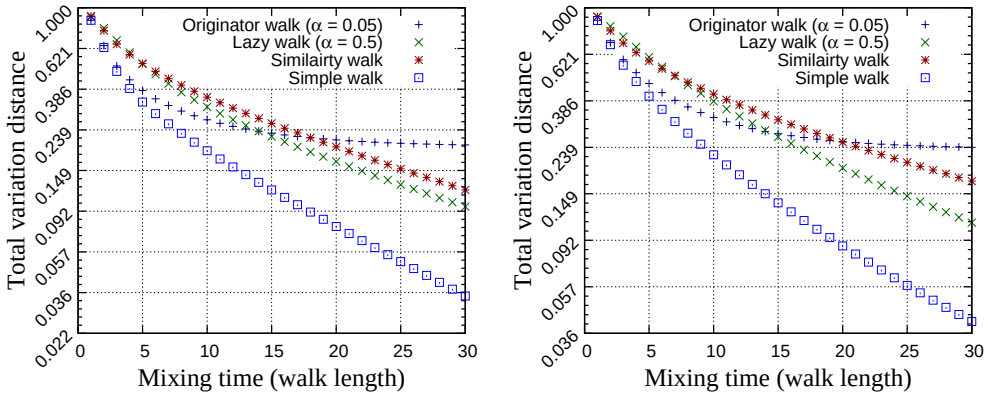

(a) Facebook A

(b) Livejournal A

(c) Facebook A

(d) Livejournal A

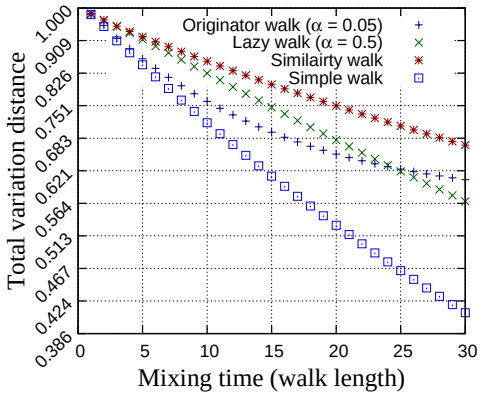
Figure 5.3: The impact of the originator and lazy walks on the mixing time—(a) and (b) are for originator-biased while (c) and (d) are for lazy random walks.



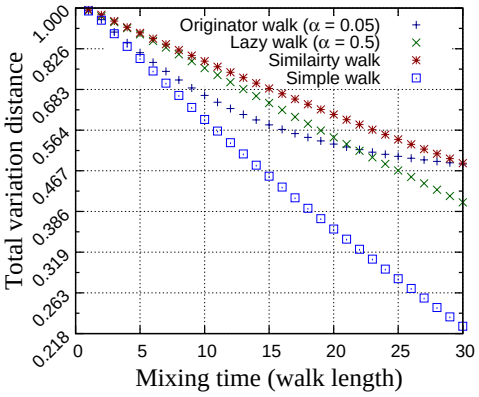

(a) Facebook A




(b) Livejournal A




(c) Physics 1




(d) DBLP

Figure 5.4: The mixing time of four different social graphs when using simple vs. lazy, originator, and similarity-biased random walks, for each graph. While they are similar in size, a mixing time (parameterized by the same $\epsilon$) is variable.

and depending on the underlying graph. However, two differences are specific to the originator-biased walk over the lazy random walk.

First, the insensitivity shown earlier is even clearer in the originator-biased model. Second, while the end result of SybilLimit operating with lazy random walk is identical to the simple random walk if one allows long enough walk to compensate for the laziness, the behavior of the originator-biased walk is different. The indirect implication of the originator-assigned probability to herself is "discontinuity" in the graph (with respect to each node), where each node gives up some of the network by not trusting nodes in it. To cover the whole graph with that same $\alpha$, $w$ needs to be exponentially large. To challenge the insensitivity of the fast mixing social graphs, we extend $\alpha$ beyond the values used in Fig. 5.6 with Facebook from Table 5.1 and use $\alpha(0 \leq \alpha \leq 0.5)$ with 0.1 steps and compute the admission rate. The result shows (not included here) that the originator-biased walk limits the number of accepted nodes, even in fast mixing graphs, but for larger $\alpha$.

#### 5.2.2.3 Performance Over Similarity and Interaction-biased Walk

The similarity and interaction-biased random walks as used in this work are unparameterized. We compute the similarity for Facebook in Table 5.1. The similarity is then used to assign weights to edges between nodes, and bias the transition matrix. We run SybilLimit with similarity-biased random walks on Facebook in Table 5.1, where the result is shown in Fig. 5.8. In short, the similarity – while expected to capture some truth about the underlying graph – has less influence on the behavior of SybilLimit. It is however worth noting that the impact of the similarity-biased random walk is clearer on other social graphs, such as DBLP and Physics, which have clearer community structures.

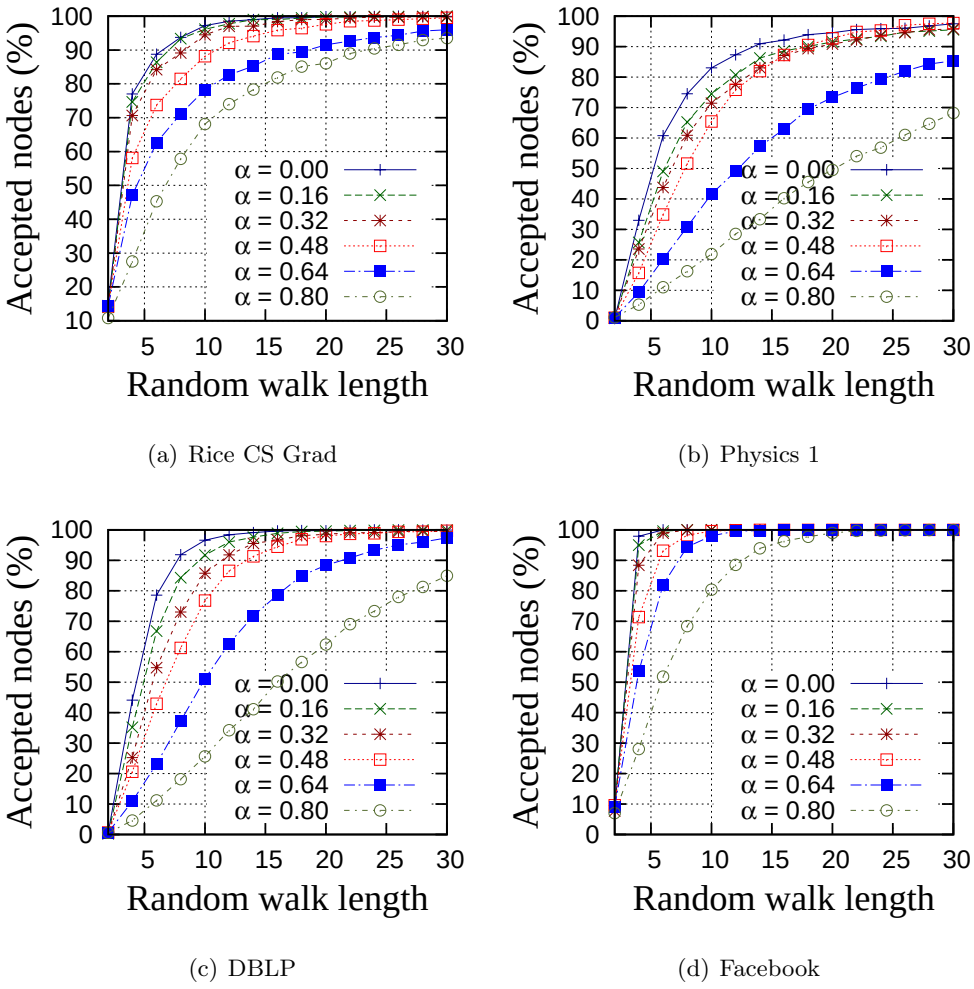
For the interaction-biased design, we borrow the interaction graph of Wilson et al. [**?**] on Facebook. The interaction model introduces a richer model than the mere connections between nodes: it shows how strong are the links between nodes in the graph. With the same settings as earlier, we run SybilLimit – as a simple random walks – over the interaction graph. The results are shown in Fig. 5.8.

## 5.3 All designs: A Comparative Study

Finally, we consider all designs at the same time. Because we only have interaction measurements for the Facebook dataset, we limit ourselves to that dataset. The result is shown in Fig. 5.8. While the performance of the similarity-biased random walk produces *almost* same results as the simple random-walk, the interaction-biased walk affects the number of the accepted nodes. Furthermore, the lazy random walk captures the behavior of model when deviated from the simple random-walk. As shown for this dataset, the interaction model behavior is characterized by the behavior of the lazy random walk for two given parameters ($\alpha = 0.48$ and $\alpha = 0.64$) suggesting that the interaction model can be further modeled as a lazy random walk where the problem is to find the proper parameters to match its behavior. Note that the value of $\alpha$ works for this dataset in particular. However, other datasets may be characterized by other values. We also find that the number of escaping tails per node is also decreased using our design, as shown in Fig. 5.9. In this last experiment, we compute the average escaping tails per 100 honest node samples, and by running the experiment 5 times, independently with a the given attackers edges for which nodes are selected uniformly at random from the honest region. In the experiment of Fig. 5.9, and for the interaction model, we assume that the attacker may infiltrate the social graph but cannot produce meaningful interactions, and thus the number of escaping tails to the attacker is always zero. It would be interesting in the future to generalize this model to an attacker with limited budget of interactions, and see how this changes the number of escaping tails with varying budgets. Finally to understand the impact of the different random walks on the accepted Sybil nodes per attack edge, we experiment with the same dataset (Facebook) and for varying $g$. The results are shown in Figure 5.10. Similar to above, our designs outperform the uniform design.

## 5.4 Implications of Findings

To sum up, we find in this study that one can control the behavior of the social network-based Sybil defenses by incorporating parameters for trust. For this purpose, we introduced and experimented the behavior of four designs. In graphs that are empirically-proven to be fast mixing and well-performing for the utility of the Sybil defense – though



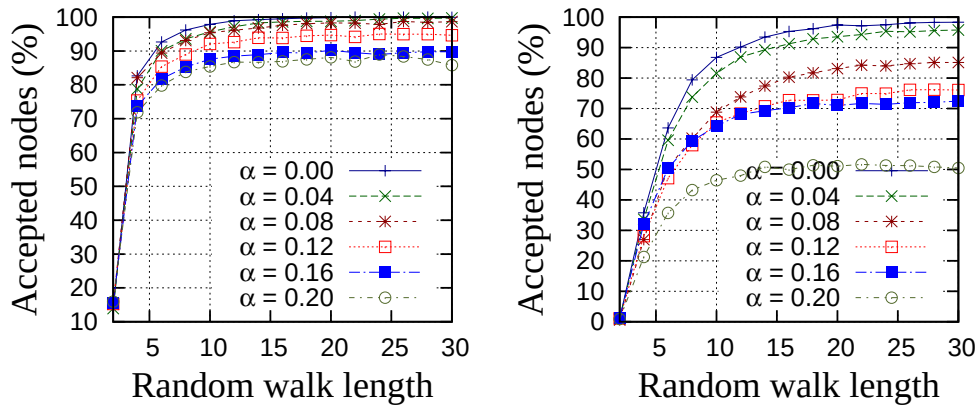

(a) Rice CS Grad

(b) Physics 1

(c) DBLP

(d) Facebook

Figure 5.5: The performance of SybilLimit measured for accepted honest nodes when using different lengths of lazy random walk for different social graphs.




(a) Rice CS Grad

(b) Physics 1

(c) DBLP

(d) Facebook

Figure 5.6: The performance of SybilLimit depends on the underlying social graph, where different graphs require different walk lengths to ensure the same number of accepted nodes. The originator-biased random walk can further influence the number of nodes accepted in each graph.

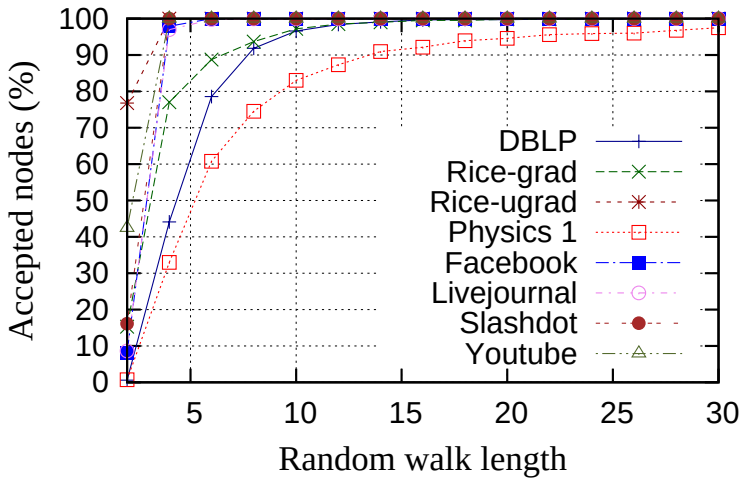



Figure 5.7: Accepted honest nodes in SybilLimit versus random walk length – with simple random walk. Different graphs have different quality of the algorithmic property though being with same size.

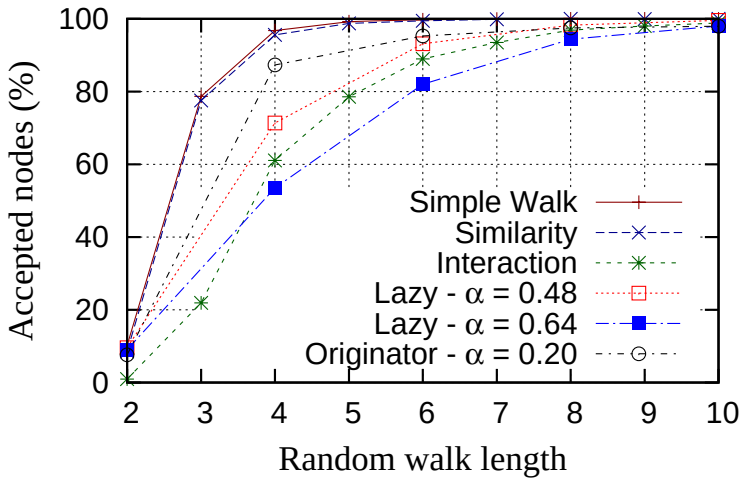



Figure 5.8: Accepted honest nodes in SybilLimit versus random walk length, when using the different designs to model the of trust in the social graph. The social graph of Facebook are sampled, where each has the size of 10,000 nodes.

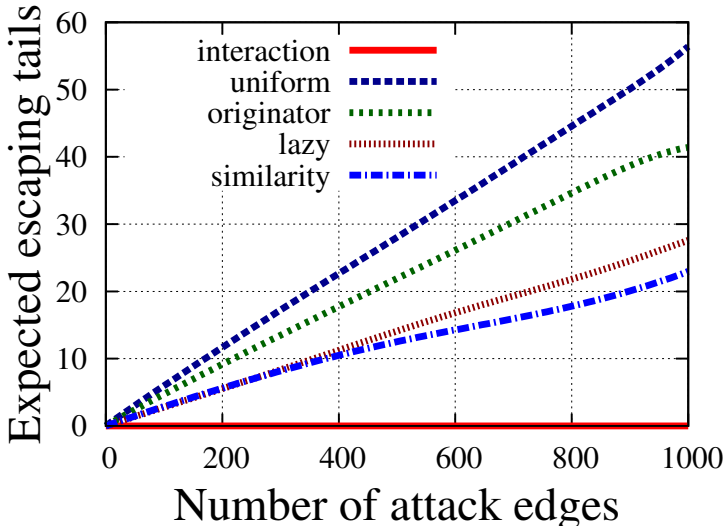



Figure 5.9: Expected escaping walks per node (among 100 nodes, $r = 850$) in Facebook dataset (in Table 5.1) where $w = 6$ and $\alpha = 0.4$ for both of the originator and lazy random walks.

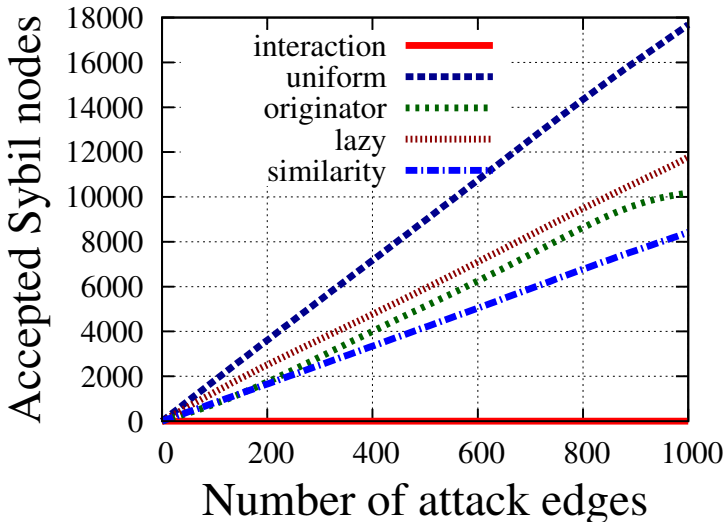



Figure 5.10: Accepted Sybil nodes over tainted tails when varying $g$in Facebook dataset (in Table 5.1) where $w = 6$ and $\alpha = 0.4$ for both of the originator and lazy random walks.

having poor value of trust – we have shown that one can select the necessary parameters to account for trust and make the performance of the defense on that graph equivalent to stronger and richer version of the same graph – e.g., the case of the interaction-based model versus the mere connections on the Facebook dataset. With these designs being intuitive in characterizing trust, the results being in agreement one another, and with this work being the first of its own type in this direction, we believe that this study is a first step in the direction of bringing well-received theoretical results into practice. The implications of our findings can be summarized as follows.

First, the mixing time and utility of the Sybil defense depend on the underlying graph. Through measurements, we supported our hypothesis that the quality of the social graph depends on the characteristic of the social links between the nodes. On one hand, social links that are easier to make result in well-enmeshed graphs but are bad in principle for the Sybil defense since they already tolerate bad edges. However, these are shown to provide good honest nodes acceptance rate even with shorter random walks. On the other hand, social links that are harder to make result in graphs with more community structure, which are bad for the detection (as shown in [**?**]) and require longer walks to operate for the honest nodes.

Second, it is now possible for the Sybil defense operator, when given multiple options of social graphs, to further derive the utility of the Sybil defense using several criteria. Our study empowers the operators by an additional dimension that influences the behavior of the Sybil defense: trust.

Third, our findings answer a recently called for question in [**?**] of studying the behavior of Sbyil defenses when operated on the interaction-based model rather than the mere social connections, which are sometimes less meaningful. In short, our study shows that the interaction model can influence the behavior of the Sybil defense, by requiring longer random walk for the defense to work for honest nodes. However, this finding also suggests that a more community-structure is in the interaction model than in the mere social graph. This implies that, while the original social graph does not possess clear community structure, the use of the interaction model would add sensitivity for the detection part of the defense and result in weaker detection. However, the underlying graphs in both cases are different and the interpretation of the results should also consider the trust value in the interaction model, which is a better fit to the trust required

in the Sybil defense.

Finally, online social graphs are known to possess weaker value of trust [**?**]. However, their potential for being used for Sybil defenses is very high since alternatives are limited, too expensive, and may not fit into the Sybil defense settings. For example, co-authorship social graphs which are known for their trust value may not necessarily include most users of a particular online system that tries to deploy the Sybil defense. On the other hand, given the popularity of online social networks, Sybil defenses may benefit from them, across systems and networks. To this end, the main finding of the work is to open the door wide for investigating trust, its modeling, and quantification for these systems.

## 5.5 Summary

In conclusion, we propose several designs to capture the trust value of social graphs in social networks used for Sybil defenses. Our designs filter weak trust links and successfully bound the mixing time which controls the number of accepted nodes using the Sybil defenses to account for variable trust. Our designs provide defense designers with parameters to model trust and evaluate Sybil defenses based on the "real value" of social networks.

Several directions are worth investigation in the near future. First, we would like to investigate generalized node-wise parameterized designs that consider different parameters for different users, or categories of them. Second, we would like to theoretically formulate the behavior of the different designs considering other features of the underlying graph, e.g., its eigenvalues, mixing time, etc. Finally, we would like to investigate the applicability of these designs in other contexts where the trust of social networks is used.

# Chapter 6

# SocialCloud: Distributed Computing on Social Networks

In this direction, we oversee a new type of computing paradigm, called SocialCloud, that enjoys parts of the merits provided by the conventional cloud. Imagine the scenario of a computing paradigm where users who collectively construct a pool of resources perform computational tasks on behalf of their social acquaintance. Our paradigm and model are similar in many aspects to the conventional grid-computing paradigm. It exhibits such similarities in that users can outsource their computational tasks to peers, complementarily to using friends for storage, which is extensively studied in literature. Our paradigm is, however, very unique in many aspects as well. Most importantly, our paradigm exploits the trust exhibited in social networks as a guarantee for the good behavior of other "workers in the system". Accordingly, the most important ingredient to our paradigm is the social bootstrapping graph, a graph that is used for recruiting workers for a social network.

## 6.1 Introduction and Preliminaries

This popularity of social networks has opened the door wide for investigating the potential of these networks for many applications. Problems that are unsolvable in the cyberspace are easily solvable using social networks, for that they possess both algorithmic properties—such as connectivity—and trust, which are used to reason about the

behavior of honest users in the social network, and limit the misbehavior introduced by other malicious users supported by efficiency features. Most important to the context of our paradigm is the aggregate computational power of nodes in the social network. Indeed, beyond the nodes and social links, the social networks consist of users with computing machines that are idle for most of the time [**?**]. Furthermore, owners of these computing machines might be willing to share their computing resources for their friends, and for a different economical model than in the conventional cloud computing paradigm—fully altruistic one. This behavior makes our work share commonalities with an existing stream of work on creating computing services through volunteers [**?**, **?**]. Our results hence highlight technical aspects of this direction and pose challenges for designs options when using social networks for recruiting such workers and enabling trust.

### 6.1.1 Contributions

To this end, our contribution in this direction is mainly:

- First, we investigate the potential of the social cloud computing paradigm by introducing a design that bootstraps from social graphs to construct distributing computing services. We advocate the merits of this paradigm over existing ones such as the grid computing paradigm.
- Second, we verify the potential of our paradigm using simulation set-up and real-world social graphs with varying social characteristics that reflect different, and possibly contradicting, trust models. Both graphs and the simulator are made public [**?**] to the community to make use of them, and improve by additional features.

### 6.1.2 The Case for SocialCloud

In this work, we look at the potential of using unstructured social graphs for building distributed computing systems. These systems are proposed with several anticipated benefits in mind. First, such systems would exploit locality of data based on the applications they are intended for, under the assumption that the data would be stored at multiple locations and shared among users represented in the social network—see §6.2.1 and [**?**] for concrete examples of such applications. This is in fact not a far-fetched

assumption. For example, consider a co-authorship social graph, like the one used in our experiments, where the SocialCloud is proposed for deployment. In that scenario, data on which computations are to be performed is likely to be at multiple locations; on machines of research collaborators, co-authors, or previous co-authors. Even for some online social networks, the assumption and achieved benefits are not far-fetched as well, considering that friends would have similar interests, and likely to have contents replicated across different machines, which could be potentially of interest to use in our computing paradigm. Examples of such settings include photos taken at parties, videos—for image processing applications, among others.

The second advantage of this paradigm is its trustworthiness. In the recent literature, there has been a lot of interest in the distributed computing community for exploiting social networks to perform trustworthy computations. Examples of these literature works include exploiting social networks for cryptographic signing services [**?**], Sybil defenses [**?**, **?**, **?**], and routing in many settings including the delay tolerant networks [**?**, **?**]. In all of these cases, along with the algorithmic property in these social networks, the built designs exploit the trust in social networks. The trust in these networks rationalizes the assumption of collaboration in these built system, and the tendency of nodes in the network to act according to the intended protocol with the theorized guarantees. Same as in all of these applications, SocialCloud tries to exploit the trust aspect of the social network, and thus it is easy to reason about the behavior of nodes in this paradigm (c.f. §6.1.4).

Related to trust exhibited in the social fabric utilized in our paradigm, the third advantage is that it is also easy to reason about the recruitment of workers. In this context, workers are nodes that are willing to perform computing tasks for other nodes (tasks outsourcers). This feature, when associated with the aforementioned trust, is quite advantageous when compared to the challenge of performing trustworthy computing on dedicated workers in the conventional grid-computing paradigm, where it is hard to recruit such workers.

Finally, our design oversees an altruistic model of SocialCloud, where nodes participate in the system and do not expect in return. Further details on this model are in §6.1.4.

**Grid Computing.** While the SocialCloud uses a similar paradigm to that of the grid

computing paradigm—in the sense that both try to outsource computations and use high aggregate computational resources, the SocialCloud is slightly different. In particular, in the SocialCloud, there is a pre-defined relationship between the task outsourcer and the computing worker, which does not exist in the grid-computing paradigm. We limit the computations to $1-$hop neighbors, which further improve trustworthiness of computations in our model.

### 6.1.3 Social Networks and Systems Bootstrapping

Social networks are so popular. Nine of the twenty most popular sites on the web are for social networking [**?**]. The top ten online social networking websites have more than 650 million of unique visitors per month in total. The most popular social network, Facebook [**?**] alone serves 250 million unique visitors per month, with more than 96 unique visitors per second. Such popularity of social networks has motivated so many designs, protocols, and applications on top of social networks. Examples include routing [**?**, **?**, **?**, **?**], social gossip [**?**, **?**, **?**], and Sybil defenses [**?**] (c.f. §6.6). While they are different in the details of their operation, all of these designs and protocols weigh algorithmic properties (connectivity), trust, and collaboration in the underlying social networks, which are used for bootstrapping such systems.

### 6.1.4 Economics of SocialCloud

In our design we assume an altruistic model, which simplifies the behavior of users and arguments on the attacker model. In this altruistic model, users in the social network *donate* their *computing resources*—while not using them—to other users in the social network to use them for specific computational tasks. In return, the same users who donated their resources for others would anticipate others as well to perform their computations on behalf of them when needed.

One can further improve this model. Social networks are rich of trust characteristics that capture additional features, and can be used to rationalize this model in several ways. For example, trust in social networks, a well studied vein of research in this context [**?**], can be used to adjust this model so as users would bind their participation in computations to trust values that they assign to other users. In this work, in order

to make use of and confirm this model, we limit outsourced computations at 1-hop.

While we do not consider that in this work (and would be a potential proposed work over the current findings), another model using interests and groups is worth mentioning for its popularity and potential as a future work. The incentives model can be further relaxed by enabling "interest" based model of computation where workers do computation to other nodes in the graph that only share some interest with them. This interest can be publicly identified by the membership of a node in a group. Investigating this model is left as a future work.

## 6.2 Use and Attack Models

### 6.2.1 Use Model and Applications

For our paradigm, we envision compute intensive applications, for which other systems have been developed in the past using different design principles, but lacking trust features; where trust is needed in such applications and provided by our paradigm. These systems include ones with resources provided by volunteers, as well as grid-like systems, like in Condor [**?**], MOON [**?**], Nebula [**?**, **?**], and SETI@Home [**?**].

Specific examples of applications built on top of these systems, that would as well fit to our use model, include blog analysis [**?**], web crawling and social-network applications (collaborative filtering, image processing, etc) [**?**], scientific computing [**?**], among others.

Notice that each of these applications requires certain levels of trust for which social ties are best suited as a trust bootstrapping and enabling tool. Especially, reasoning about the behavior of systems and expected outcomes (in a computing system in particular) would be well-served by this trust model. We notice that this social trust has been previously used as an enabler for privacy in file-sharing systems [**?**], anonymity in communications systems [**?**], and collaboration in sybil defenses [**?**, **?**, **?**], among others. In this work, we use the same insight to propose a computing paradigm that relies on such trust and volunteered resources, in the form of shared computing time. With that in mind, in the following section we elaborate on the attacker used in our system and trust models provided by our design, thus highlight its advantage and distancing our work from prior works in the literature.

### 6.2.2 Attacker Model

In this work, as it is the case in many other systems built on top of social networks [**?**, **?**, **?**], we assume that the attacker is restricted in many aspects. For example, the attacker has a limited capability of creating arbitrarily many edges between himself and other nodes in the social graph.

While this restriction may contradict some recent results in the literature [**?**]—where it is shown that some legitimate users befriend random users in the social network who are potentially attackers, it can be relaxed to achieved the intended trust and attack model by considering an overlay of subset of friends of each users. This overlay expresses the trust value of the social graph well and eliminates the influence introduced by the attacker who infiltrated the social graph [**?**]. For example, since each user decides on to which node among his adjacent nodes to outsource computations to, each user is aware of other users he knows well and those who are just social encounters that could be potential attackers. Accordingly, the user himself decides whether to include a given node in his overlay or not, thus minimizing or eliminating harm and achieving the required trust and attack model.

The description of the above attacker model might be at odds with the rest of the work, especially that we use some online social networks that do not reflect characteristics of trust required in our paradigm. However, such networks, when used, are used for two reasons. First, to derive insight on the potential of such social networks, and others that share similar topological characteristics, for performing computational tasks according to the method devised in this work. Second, we use them to illustrate that some of these social networks might be less effective than the trust-possessing social graphs, which we strongly advocate for our computing paradigm.

**Comparison with Trust in Grid Computing Systems.** While there has been a lot of research on characterizing and improving trust in the conventional grid computing paradigm [**?**, **?**, **?**, **?**]—which is the closest paradigm to compare to ours, trust guarantees in such paradigm are less strict than what is expressed by social trust. For that, it is easy to see that some nodes in the grid computing paradigm may act maliciously by, for example, giving wrong computations, or refusing to collaborate; which is even easier to detect and tolerate, as opposed to acting maliciously [**?**].

## 6.3 The Design of SocialCloud

The main design of SocialCloud is very simple, where complexities are hidden in design choices and options. In SocialCloud, the computing overlay is bootstrapped by the underlying social structure. Accordingly, nodes in the social graph act as workers to their adjacent nodes (i.e., nodes which are one hop away from the outsourcer of computations). An illustration of this design is depicted in Figure 6.1. In this design, nodes in the social graph, and those in the SocialCloud overlay, use their neighbors to outsource computational tasks to them. For that purpose, they utilize local information to decide on the way they schedule the amount of computations they want each and every one of their neighbors to take care of. Accordingly, each node has a scheduler which she uses for deciding the proportion of tasks that a node wants to outsource to any given worker among her neighbors. Once a task is outsourced to the given worker, and assuming that both data and code for processing the task are transferred to the worker, the worker is left to decide how to schedule the task locally to compute it. Upon completion of a task, the worker sends back the computations result to the outsourcer.

### 6.3.1 Design Options: Scheduling Entity

In the SocialCloud, two schedulers are used. The first scheduler is used for determining the proportion of task outsourced to each worker and the second scheduler is used at each worker to determine how tasks outsourced by outsourcers are computed and in which order. While the latter scheduler can be easily implemented locally without impacting the system complexity, the decision used for whether to centralize or decentralize the former scheduler impacts the complexity and operation of the entire system. In the following, we elaborate on both design decisions, their characteristics, and compare them.

- **Decentralized scheduler.** In our paradigm, we limit selection of workers to 1-hop from the outsourcer. This makes it possible, and perhaps plausible, to incorporate scheduling of outsourcing tasks at the side of the outsourcer in a decentralized manner—thus each node takes care of scheduling its tasks. On the one hand, this could reduce the complexity of the design by eliminating the scheduling server in a centralized alternative. However, on the other hand, this could

increase the complexity of the used protocols and the cost associated with them for exchanging *states*—such as availability of resources, online and offline time, among others. All of such states are exchanged between workers and outsourcers in our paradigm. These states are essential for building basic primitives in any distributed computing system to improve efficiency (see below for further details). An illustration of this design option is shown in Figure 6.1. In this scenario, each outsourcer, as well as worker, has its own separate scheduling component.

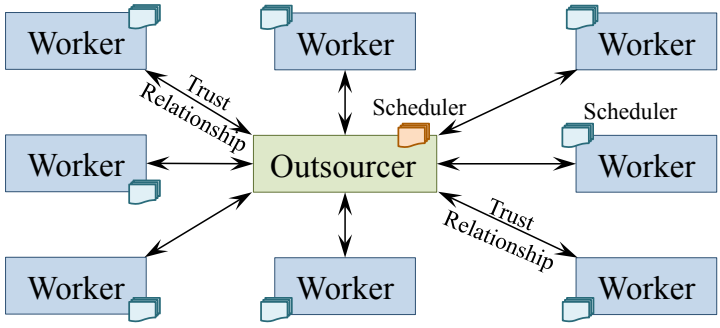



Figure 6.1: A depiction of the main SocialCloud paradigm as viewed by an outsourcer of computations. The different nodes in the social network act as workers for their friends, who act as potential jobs/tasks outsourcers. The links between social nodes are ideally governed by a strong trust relationship, which is the main source of trust for the constructed computing overlay. Both job outsourcers and workers have their own, and potentially different, schedulers.

- **Centralized Scheduler.** Despite the fact that nodes may only require their neighbors to perform the computational tasks on behalf of them and that may require only local information—which could be available to these nodes in advance, the use of a centralized scheduler might be necessitated to reduce communication overhead at the protocol level. For example, in order to decide upon the best set of nodes to which to outsource computations, a node needs to know which of its neighbors are available, among other statistics. For that purpose, and given that the underlying communication network topology may not necessarily have the same proximity of the social network topology, the protocol among nodes needs to incur back and forth communication cost. One possible solution to the problem

is to use a centralized server that maintains states of the different nodes. Instead of communicating directly with neighbor nodes, an outsourcer would request the best set of candidates among its neighbors to the centralized scheduling server. In response, the server will produce a set of candidates, based on the locally stored states. Such candidates would typically be those that would have the most available resources to handle the outsourced computation task.

An illustration of this design option is shown in Figure 6.2. In this design, each node in SocialCloud would periodically send states to a centralized server. When needed, an outsourcer node contacts the centralized server to return to it the best set of candidates for outsourcing computations, which the server would return based on the states of these candidates. Notice that only states are returned to the outsourcer, upon which the outsourcer would send tasks to these nodes on its own—Thus, the server involvement is limited to the control protocol.

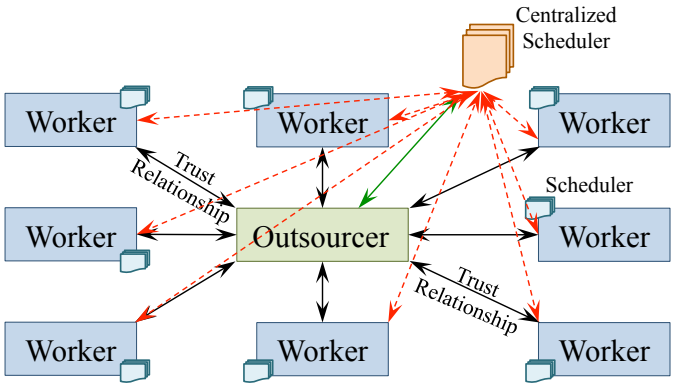



Figure 6.2: The decentralized model of task scheduling in SocialCloud.

The communication overhead of this design option to transfer states between a set of $d$ nodes is $2d$, where $d$ messages are required to deliver all nodes' states and $d$ messages are required to deliver states of all other nodes to each node in the set. On the other hand, $d(d-1)$ messages are required in the decentralized option (which requires pair-wise communication of states update). When outsourcing of computations is possible among all nodes in the graph, this translates into $O(n)$ for the centralized versus $O(n^2)$ communication overhead for the decentralized option. To sum up, Table 6.1 shows a

comparison between both options.

Table 6.1: A comparison between the centralized and decentralized scheduler options. Compared features are resistance to failure, communication overhead, required additional hardware, and required additional trust.

| Option | Failure | Communication | Hardware | Trust |
|---|---|---|---|---|
| Centralized | ✖ | $O(n)$ | ✖ | ✖ |
| Decentralized | ✔ | $O(n^2)$ | ✔ | ✔ |

### 6.3.2 Tasks Scheduling Policy

While the use of distributed or centralized scheduling entity resolves the issue of scheduling at the outsourcer side, two decisions remain unsolved: how much computation to outsource to each node (worker), and how much time a node among these workers should spend on a given task for a certain outsourcer. We handle these two issues separately.

As mentioned earlier, any off-the-shelf scheduling algorithm can be utilized to decide the right scheduling policy at the side of the outsourcer, which can be further improved by incorporating trust characterization models for weighted job scheduling [**?**]. On the other hand, for workers scheduling, we consider several scheduling options as follows (notice that all of these policies are applied with respect to "computing time". This further requires estimating the time required for each task as a first step for using these policies).

- **Round Robin (RR) Scheduling Policy.** This is the simplest policy to implement, in which a worker spends an equal share of time on each outsourced task in a round robin fashion among all tasks he has.
- **Shortest First (SF) Scheduling Policy.** The worker performs shortest task first.
- **Longest First (LF) Scheduling Policy.** The worker performs longest task first.

Notice that we omit a lot of details about the underlying computing infrastructure, and abstract such infrastructure to "time sharing machines", which further simplifies much of the analysis in this work. In the results, we experiment with the three scheduling policies.

### 6.3.3 Handling Outliers

The main performance criterion used for evaluating SocialCloud is the time required to finish computing tasks for all nodes with tasks in the system. Accordingly, an outlier (also called a computing straggler) is a node with computational tasks that take a long time to finish, thus increasing the overall time to finish and decreasing the performance of the overall system. Detecting outliers in our system is simple: since the total time is given in advance, outliers are nodes with computing tasks that have longer time to finish when other nodes participating in the same outsourced computation are idle. Our method for handling outliers is simple too: when an outlier is detected, we outsource the remaining part of computations on all idle nodes neighboring the original outsourcer. For that, we use the same scheduling policy used by the outsourcer when she first outsourced this task. In the simulation part, we consider both scenarios of handled and unhandled outliers, and observe how they affect the performance of the system.

### 6.3.4 Deciding Workers Based on Resources

In real-world deployment of a system like SocialCloud, we expect heterogeneity of resources, such as bandwidth, storage, and computing power, in workers. This heterogeneity would result in different results and utilization statistics of a system like SocialCloud, depending on which nodes are used for what tasks. While our work does not address this issue, and leaves it as a future work. We further believe that simple decisions can be made in this regard so as to meet the design goals and achieve the good performance. For example, we expect that nodes would select workers among their social neighbors that have resources and link capacities exceeding a threshold, thus meeting an expected performance outcome.

## 6.4 Simulator of SocialCloud

To demonstrate the potential of SocialCloud as a computing paradigm, we implement a batch-based simulator [**?**] that considers a variety of scheduling algorithms, an outlier handling mechanism, job generation handling, and failure simulation. A flow diagram of the simulator is in Figure 6.3.

### 6.4.1 Flow Diagram of the Simulator

The flow of the simulator, which represents the flow of the system, is depicted in Figure 6.3. First, the node factory uses the bootstrapping social graph to create nodes and their workers. Each node then decides on whether she has a task or not, and if she has a task she schedule the task according to her scheduling algorithm. If needed, each node then transfers code on which computations are to be performed to the worker along with the splits of the data for these codes to run on. Each worker then performs the computation according to the scheduling algorithm of the worker and returns the results of the computations to the outsourcer.

### 6.4.2 Timing

In SocialCloud, we use *virtual time* to simulate computations and resources sharing. We scale down the simulated time by 3 orders of magnitude of that in reality. This is, for every second worth of computations in real-world, we use one millisecond in the simulation environment. Thus, units of times in the rest of this work are in virtual seconds.

### 6.4.3 Settings

In this section, in order to derive insight on the potential of SocialCloud, we experiment with the simulator described above. Before getting into the details of the experiments, we describe the data and evaluation metric used in this section.

#### 6.4.3.1 Evaluation Metric

To demonstrate the potential of operating SocialCloud, we use the "normalized finishing time" of a task outsourced by a user to other nodes in the SocialCloud as the performance metric. We consider the same metric over the different graphs used in the simulation. To demonstrate the performance for the population of all nodes that have tasks to be computed in the system, we use the empirical CDF (commutative distribution function) as an aggregate measure. For a random variable $X$, the CDF is defined as $F_X(x) = P_r(X \leq x)$. In our experiments, the CDF measures the fraction (or percent) of nodes that finish their tasks before a point in time $x$, as part of the overall number of tasks. We define $x$ as the factors of time of normal operation per dedicated machines, if they were to be used instead of outsourcing computations. This is, suppose that the overall time of a task is $T_{tot}$ and the time it takes to compute the subtask by the slowest worker is $T_{last}$, then $x$ for that node is defined as $T_{last}/T_{tot}$.

#### 6.4.3.2 Tasks Generation

Also for demonstrating the operation of our simulator, and the trade-off that such operation provides, we consider two different approaches for the tasks generated by each user. The size of each generated task is measured by virtual units of time, and for our demonstration we use two different scenarios:

- **Constant task weight.** each outsourcer generates tasks with an equal size. These tasks are divided into equal shares and distributed among different workers in the computing system. The size of each task is $\bar{T}$.
- **Variable task weight.** each outsourcer has a different task size. We model the size of tasks as a uniformly distributed random variable in the range of $[\bar{T} - \ell, \bar{T} + \ell]$ for some $\bar{T} > \ell$. Each worker receives an equal share of the task from the outsourcer.

#### 6.4.3.3 Deciding Tasks Outsourcers

Not all nodes in the system are likely to have tasks to outsource for computation at the same time. Accordingly, we denote the fraction of nodes that have tasks to compute

by $p$, where $0 < p < 1$. In our experiments we use $p$ from 0.1 to 0.5 with increments of 0.1. We further consider that each node in the network has a task to compute with probability $p$, and has no task with probability $1 - p$—thus, whether a node has a task to distribute among its neighbors and compute or not follows a binomial distribution with a parameter $p$. Once a node is determined to be among nodes with tasks at the current round of run of the simulator, we fix the task length. For tasks length, we use both scenarios mentioned in §6.4.3.2; with fixed or constant and variable tasks weights.

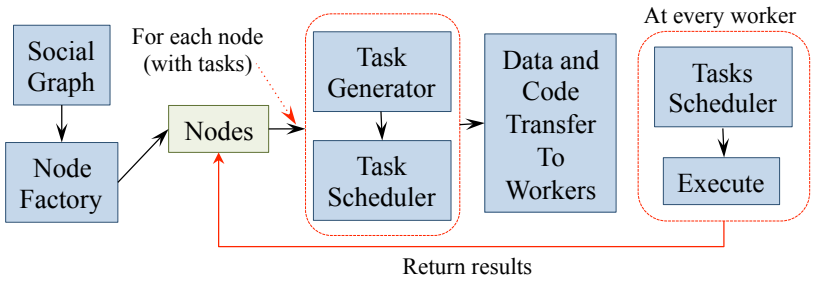



Figure 6.3: The flow diagram of SocialCloud: social graph is used for bootstrapping the computing service and recruit workers, nodes are responsible for scheduling their tasks by determining the amount of work each of its neighbors would process, and each worker (node) uses its local scheduler to determine how much time is allowed for each sub-task by its neighbors.

## 6.5 Results and Measurements

In this section we demonstrate our paradigm and discuss the main results of this work. Due to the lack of space, we delegate additional results to the technical report in [**?**]. For all measurements, our metric of performance and comparison is the normalized time to finish metric, explained in section 6.4.3.1.

### 6.5.1 Social Graphs

To derive insight on the potential of SocialCloud, we run our simulator on several social graphs with different size and density as shown in Table 6.2. The graphs used in these experiments represent three co-authorship social structures (DBLP, Physics 1, and

Physics 2), one voting network (of Wiki-vote for wikipedia administrators election), and one friendship network (of the consumer review website, Epinion). All of these graphs are made undirected, if they are not already, which rationalizes their use in our system. Notice the varying density of these graphs, which also reflects on varying topological characteristics. Also, notice the nature of these social graphs, where they are built in different social contexts and possess varying qualities of trust [**?**].

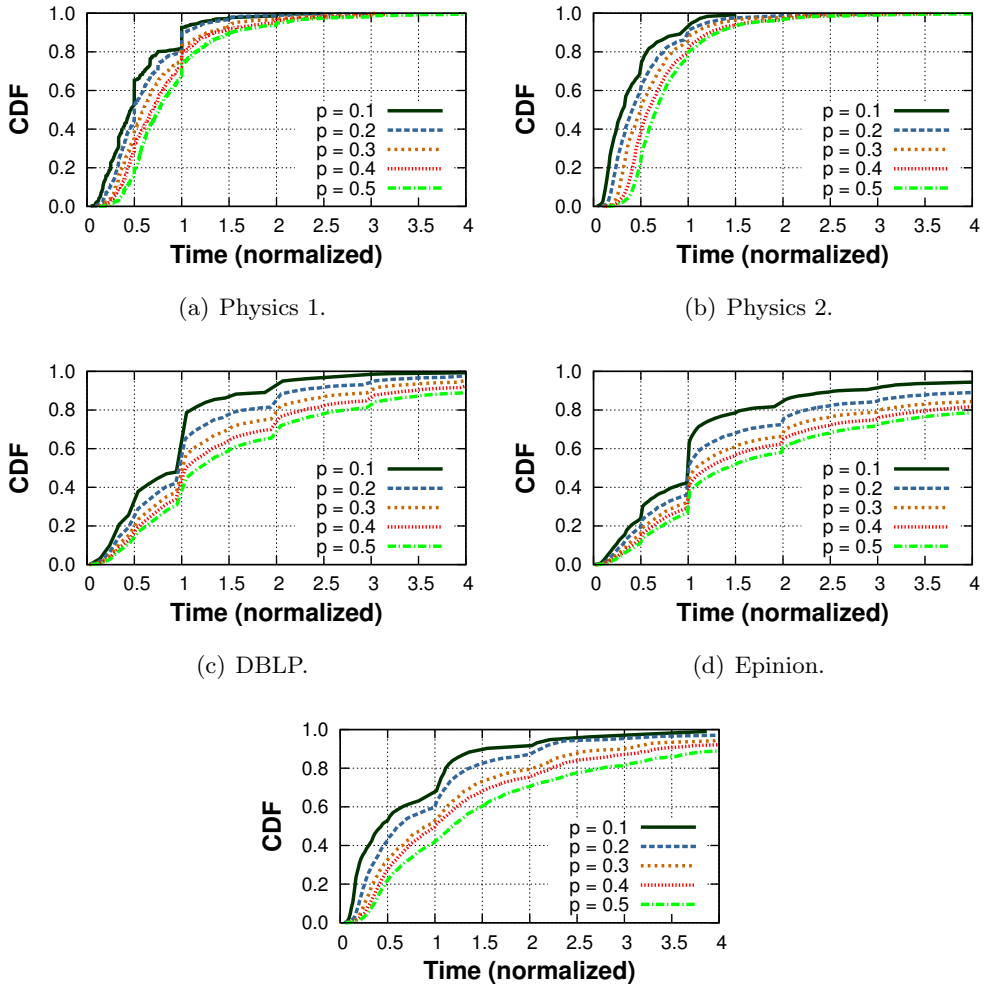
Table 6.2: Social graphs used in our SocialCloud simulator.

| Dataset | # nodes | # edges | Description |
|---|---|---|---|
| DBLP | 614981 | 1155148 | CS Co-authorship |
| Epinion | 75877 | 405739 | Friendship network |
| Physics 2 | 11204 | 117649 | Co-authorship |
| Wiki-vote | 7066 | 100736 | Voting network |
| Physics 1 | 4158 | 13428 | Co-authorship |

### 6.5.2 Performance When Varying the Number of Outsourcers

In the first experiment, we run our SocialCloud simulator on the different social graphs discussed earlier to measure the evaluation metric when the number of the outsourcers of tasks increases. We consider $p = 0.1$ to 0.5 with increments of 0.1 at each time. The results of this experiment are in Figure 6.4. On the results of this experiment we make several observations.

First, we observe the potential of SocialCloud, even when the number of outsourcers of computations in the social network is as high as 50% of the total number of nodes, which translates into a small normalized time to finish even in the worst performing social graphs (about 60% of all nodes with tasks would finish in 2 normalized time units). However, this advantage varies for different graphs: we observe that sparse graphs, like co-authorship graphs, generally outperform other graphs used in the experiments (by observing the tendency in the performance in figures 6.5(b) through 6.4(c) versus figures 6.4(d) and 6.4(e)). In the aforementioned graphs, for example, we see that when 10% of nodes in each case is used, and by fixing $x$, the normalized time, to 1, the difference of performance is about 30%. This difference of performance is observed between




(a) Physics 1.

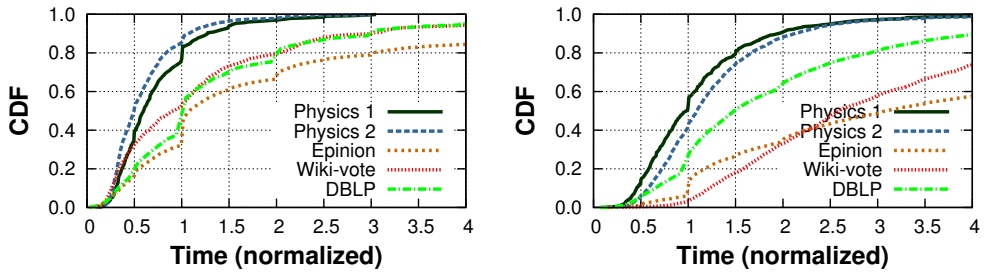
(b) Physics 2.

(c) DBLP.

(d) Epinion.

(e) Wiki-vote.

Figure 6.4: The normalized time it takes to perform outsourced computations in SocialCloud. Different graphs with different social characteristics have different performance results, where those with well-defined social structures have self-load-balancing features, in general. These measurements are taken with round-robin scheduling algorithm that uses the outlier handling policy in §6.3.3 for a fixed task size (of 1000 simulation time units).




(a) Handled outliers

(b) Unhandled outliers

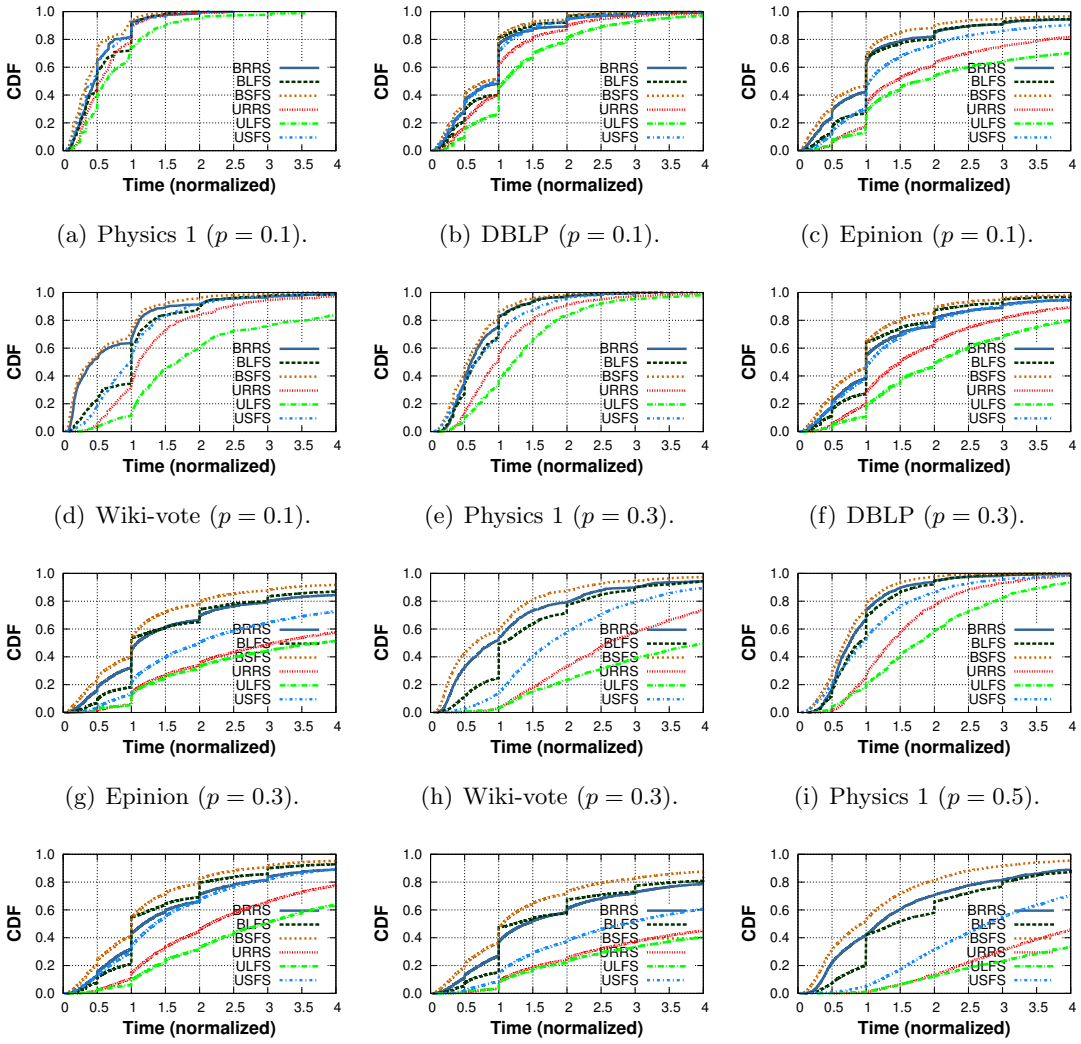
Figure 6.5: The performance of SocialCloud on the different social graphs used for our experiments, demonstrating the inherent differences in the different social graphs. Both figures use $p = 0.3$ and the round robin scheduling algorithm. Left figure is when handling outliers, whereas the right figure without handling the outliers.

the Physics co-authorship graphs—where 95% of nodes finish their computations—and the Epinion graph—where only about 65% of nodes finish their computations.

Second, we observe that the impact of $p$, the fraction of nodes with tasks in the system, would depend on the graph rather than $p$ alone. For example, in Figure 6.5(b), we observe that moving from $p = 0.1$ to $p = 0.5$ (when $x = 1$) leads to a decrease in the fraction of nodes that finish their computations from 95% to about 75%. On the other hand, for the same settings, this would lead to a decrease from about 80% to 40%, a decrease from about 65% to 30%, and a decrease from 70% to 30% in DBLP, Epinion, and Wiki-vote, respectively. This suggests that the decreases in the performance are due to an inherit property of each graph. The inherit property of each graph and how it affects the performance of SocialCloud is further illustrated in Figure 6.5. Interestingly, we find that even if DBLP is almost two orders of magnitude the size of Wiki-vote, for example, it outperforms Wiki-vote when not using outlier handling, and gives almost the same performance when using outliers handling.

### 6.5.3 Performance with Different Scheduling Policies

Now, we turn our attention to understanding the impact of the different scheduling policies discussed in §6.3.2 on the performance of SocialCloud. We consider the different




(a) Physics 1 ($p = 0.1$). (b) DBLP ($p = 0.1$). (c) Epinion ($p = 0.1$).

(d) Wiki-vote ($p = 0.1$). (e) Physics 1 ($p = 0.3$). (f) DBLP ($p = 0.3$).

(g) Epinion ($p = 0.3$). (h) Wiki-vote ($p = 0.3$). (i) Physics 1 ($p = 0.5$).

(j) DBLP ($p = 0.5$). (k) Epinion ($p = 0.5$). (l) Wiki-vote ($p = 0.5$).

Figure 6.6: The normalized time it takes to perform outsourced computations in SocialCloud for different scheduling policies. Naming convention: U stands for unhandled outlier and B stands for handled outliers (Balanced). RRS, SFS, and LFS stand for round-robin, shortest first, and longest first scheduling.

datasets, and use $p = 0.1$ to 0.5 with 0.2 increments (the results are shown in Figure 6.6). The observed consistent pattern in almost all figures in this experiment tells that shortest first policy always outperforms the round robin scheduling policy, whereas the round robin scheduling policy outperforms the longest first. This pattern is consistent regardless of $p$ and the outlier handling policy. The difference in the performance when using different policies can be as low as 2% (when $p = 0.1$ in physics co-authorship; shown in figure 6.7(b)) and as high as 70% (when using $p = 0.5$ and outlier handling as in wiki-vote (figure 6.6(l))). The patterns are made clearer in Figure 6.6 by observing combinations of parameters and policies.

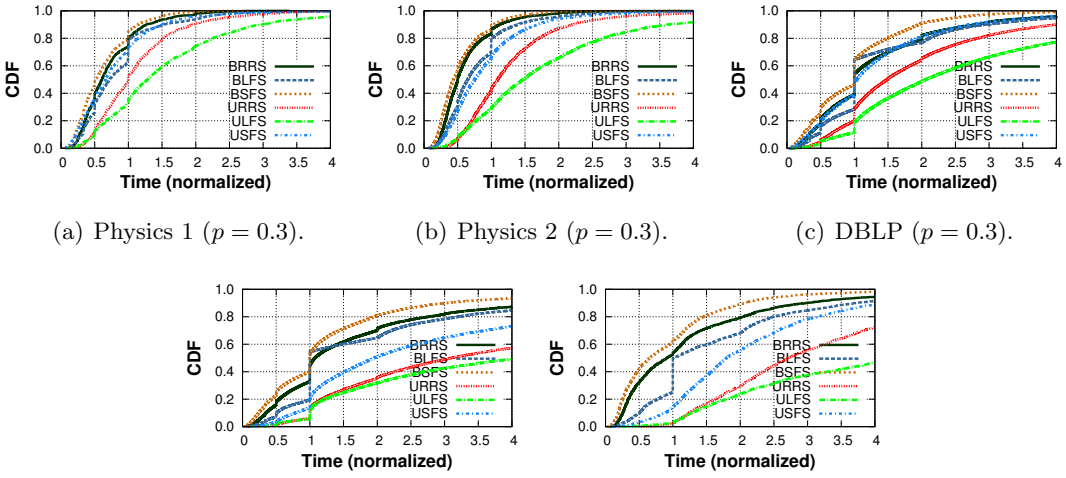
### 6.5.4 Performance with Outliers Handling

Outliers, as defined in §6.3.3, drag the performance of the entire system down. However, as pointed out earlier, handling outliers is quite simple in SocialCloud if accurate timing is used in the system. Here we consider the impact of the outlier handling policy explained in §6.3.3. The impact of using the outlier handling policy can be also seen on figure 6.6, which is used for demonstrating the impact of using different scheduling policies as well. In this figure, we see that the simple handling policy we proposed improves the performance of the system greatly in all cases. The improvement differs depending on other parameters, such as $p$, and the scheduling policy. As with the scheduling policy, the improvement can be as low as 2% and as high as more than 60%. When $p$ is large, the potential for improvement is high—see, for example, $p = 5$ in Physics 2 with the round robin scheduling policy where almost 65% improvement is due to outlier handling when $x = 1$.

### 6.5.5 Performance with Variable Task Size

In all of the above experiments, we considered computational tasks of fixed size; 1000 of virtual time units in each of them. Whether the same pattern would be observed in tasks with variable size is unclear. Here we experimentally address this concern by using variable duty size that is uniformly distributed in the interval of $[500, 1500]$ time units. The results are shown in Figure 6.7. Comparing these results to the middle row of Figure 6.6 (for the fixed size tasks), we make two observations. (i) While the average

task size in both scenarios is same, we observe that the performance with variable task size is worse. This performance is anticipated as our measure of performance is the time to finish that would be definitely increased as some tasks with longer time to finish are added. (ii) The same patterns advantaging a given scheduling policy on another are maintained as in earlier with fixed task length.




(a) Physics 1 ($p = 0.3$). (b) Physics 2 ($p = 0.3$). (c) DBLP ($p = 0.3$).

(d) Epinion ($p = 0.3$). (e) Wiki-vote ($p = 0.3$).

Figure 6.7: The normalized time it takes to perform outsourced computations in SocialCloud, for variable task size.

### 6.5.6 Relationship Between Structure and Performance

It is worth noting that the performance of SocialCloud is quite related to the underlying structure of the social graph. For example, sparse graphs such as co-authorship graphs—which are pointed out in [**?**] to be slow mixing graphs—are the graphs with performance advantage in SocialCloud. These graphs, in particular, are shown to possess a nice trust value that can be further utilized for SocialCloud. Furthermore, this trust value is unlikely to be found in online social networks which are prone to infiltration, making the case for trust-possessing graphs even stronger, as they achieve performance guarantees as well. This, indeed, is an interesting finding by itself, since it shows opposite outcomes to what is known in the literature on the usefulness of these graphs.

### 6.5.7 Additional Features and Limitations of Experiments

Our simulator of SocialCloud omits a few details concerning the way a distributed system behaves in reality. In particular, our measurements do not report on or experiment with failure. However, our simulator is equipped with functionality for handling failure in the same way used for handling outliers (c.f. §6.3.3). Furthermore, our simulator considers a simplistic scenario of study by abstracting the hardware infrastructure, and does not consider additional resources consumed, such as memory and I/O resources. In the future, we will consider equipping our simulator with such functionalities and see how this affects the behavior and benefits of SocialCloud.

One last concern related to our demonstration of our paradigm is that we do not consider the heterogeneity of resources, such as bandwidth and resources, in nodes acting as workers in the system. Furthermore, we did not consider how this affects the usability of our system and what decision choices this particular aspect of distributed computing systems would have on the utility of our paradigm. While this would be mainly a future work to consider, we expect that nodes would select workers among their social neighbors that have resources and link capacities exceeding a threshold, thus meeting an expected performance outcome.

## 6.6 Related Work

There have been many works on the use of social networks for building communication and security systems, studying the performance of such designs on top of social networks, and analyzing the assumptions used in these designs as well. Below we highlight a few examples of these efforts and works.

Systems built on top of social networks include file sharing systems [**?**], anonymous communication systems [**?**, **?**] Sybil defenses [**?**, **?**, **?**, **?**], referral and filtering systems [**?**, **?**], and live streaming [**?**]. Most of these applications weigh the trust in social graph, and an algorithmic property that makes the operation of these systems on top of social network effective. Another set of applications that exploit social networks' trust is routing [**?**, **?**, **?**, **?**]—in several settings, where it has been shown that connectivity in social graphs can be of benefit in disconnected networks. Finally, assumptions of social network-based systems are explored recently, where Sybil defenses and their assumptions

are studied in [?], and trust is challenged in [?].

Perhaps the closest vein of related work in the literature to our work is on the use of social networks for building computing services. Until the time of writing this work, most of the prior research work has been solely focused on providing storage services, but not a platform of computations. All of these systems share similar motivations as in our systems; they bring trust from an orthogonal context to empower a distributed computing system. In many cases, such storage services use slightly different economical model than the one used in SocialCloud, where payment per Megabyte per month rates are used as opposed to our eco-system. Examples of such efforts are reported by Tran et al. [?]), Chard et al. [?], and Sato [?]. Xu et al. [?] have further explored a first step in the direction of building cloud computing platforms on top of social networks by considering the access control model in this domain with preferred access control guarantees. The results of this work can be used as a building block in our work to improve the quality of access control and authorization.

Concurrent to our work, and following their work in [?], Chard et al. [?] suggested the use of social networks to build a resource sharing system. Whereas their main realization was still a social storage system as in [?], they also suggested that the same vision can be used to build a distributed computing service as we advocate in this work. Recent realizations of this vision have been reported in [?] and [?]. In [?], Thaufeeg et al. devised an architecture where "individuals or institutions contribute the capacity of their computing resources by means of virtual machines leased through the social network". In [?] Koshy et al. further explored the motivations of users to enable social cloud systems for scientific computing.

With similar flavor of distributed computing services design, there has been prior works in literature on using volunteers' resources for computations exploiting locality of data [?, ?], examination of programing paradigms, like MapReduce [?] on such paradigm [?, ?]. Finally, our work shares several commonalities with the grid and volunteer computing systems [?, ?, ?, ?, ?], of which many aspects are explored in the literature. Trust of grid computing and volunteer-based systems is explored in [?, ?, ?, ?, ?]. Applications built on top of these systems, that would fit to our use model, are reported in [?, ?, ?], among others.

## 6.7 Summary

In this work we have introduced the design of SocialCloud, a distributed computing service that recruits computing workers from friends in social networks and use such social networks that characterize trust relationships to bootstrap trust in the proposed computing service. We further advocated the case of such computing paradigm for the several advantages it provides. To demonstrate the potential of our proposed design, we used several real-world social graphs to bootstrap the proposed service and demonstrated that majority of nodes in most cases would benefit computationally from outsourcing their computations to such service. We considered several basic distributed system characteristics and features, such as outlier handling, scheduling decisions, and scheduler design, and show advantages in each of these features and options when used in our system. To the best of our knowledge, this is the first and only work in literature that bases such design of computing paradigm on volunteers recruited from social networks and tries to bring the trust factor from these networks and use it in such systems. This characteristic distances our work from the prior work in literature that uses volunteers' resources for computations [**?**, **?**].

Most important outcome of this study, along with the proposed design, is the relationship exposed between the social graphs and the behavior of the built computing service on top of them. In particular, we have shown that social graphs that possess strong trust characteristics as evidenced by face-to-face interaction [**?**], which are known in the literature for their poor characteristics prohibiting their use in applications (such as Sybil defenses [**?**, **?**, **?**]), have a self-load-balancing characteristics when the number of outsourcers are relatively small (say 10 to 20 percent of the overall population on nodes in the computing services). That is, the time it takes to finish tasks originated by a given fraction of nodes in such graph, and for the majority of these nodes, ends in a relatively short time. On the other hand, such characteristics and advantages are maintained even when the number of outsourcers of computations is as high as 50% of the nodes, contrary to the case of other graphs with dense structure and high connectivity known to be proper for the aforementioned applications. This last observation encourages us to investigate further scenarios of deployment of our design. We anticipate interesting findings based on the inherit structure of such deployment contexts—since

such contexts may have different social structures that would affect the utility of the built computing overlay.

# Chapter 7

# DynaMix: Anonymity on Dynamic Social Networks

All of the prior works which use social networks for building applications to solve real-world problems by exploiting these networks' structure and trust make certain assumptions. Some of these assumptions are rational and others are irrational in light of our understanding of the evolving settings of social networks. For example, the aforementioned social network-based Sybil defenses assume relatively simple trust model of binary relationships among nodes: a node either knows other nodes in the social graph or does not know them at all [**?**, **?**, **?**]. While simple and easy to implement, this model turns problematic when it does not represent reality by not characterizing the richer nature of social graphs that involves "differential trust" [**?**]. The idea has been recently used, and several designs have been considered to characterize trust for the problem in hand, and shown that differential trust can improve the security of Sybil defenses, despite degradation of algorithmic properties imposed by the differential trust [**?**].

Another simplifying assumption is the nature of associations among nodes; graphs are assumed to be static [**?**, **?**, **?**, **?**, **?**]. Insight is brought on the potential of these designs by experimenting with static social graphs, and by ignoring the dynamic nature of social graphs. Ignoring this nature might be due to unavailability of tools to capture the dynamic nature of social graphs, or unavailability of measures to quantify the performance of these designs on such dynamic social graphs. However, the limited nature

of the static social graphs prohibits us from getting insight of these designs in reality when applied to real-world deployment settings for which social graphs are well-known for their dynamic behavior [**?**, **?**, **?**]. Such behavior greatly alters graphs structure, which is an essential determining part of the performance of these designs.

In this work we propose to proceed further to understand dynamic social graphs for another family of applications, anonymous communication systems. On the one hand, we would like to extend and utilize earlier findings in [**?**] and [**?**] of using social graphs as good mixers for anonymity. On the other hand, we want to improve on these results by formalizing the use of dynamic social structures for anonymity, and establishing a relationship between dynamic and weighted graphs. We want to show how our new paradigm enables anonymous communication and stands against possible attacks by empowering a richer social structure. We validate our model using empirical studies on two dynamic social structures driven from real-world social networks. To this end, our contributions are as follows:

- We formally define the problem of anonymously communicating on *dynamic* social structures that are natural in many contexts. We provide tools to quantify anonymity when such structures are used. In particular, we show an interesting theoretical connection between dynamic graphs and unweighted graphs representing history of associations among nodes and influencing the way these nodes are selected by their neighbors for anonymity.

- Using our model of dynamic structures, we provide detailed and extensive experiments to show the usefulness of our proposed model in reality considering two real-world network traces that exhibit dynamic structures.

## 7.1 Preliminaries

In this section we review the preliminaries of known literature on the problem, which are required for understanding the rest of this chapter. This known literature assumes a static graph. Unless otherwise is mentioned, this formalism follows from [**?**], which is to the best of our knowledge the only work that directly touches upon the problem.

### 7.1.1 System Settings and Application Scenario

The idea of mixers over social links is very simple: users recruit their social acquaintance to provide anonymity to their traffic. In the nutshell, each node (user) forwards her own traffic to her friends, and friends forward that traffic to their friends, and so on, for a certain number of hops, say $\ell$. The number of hops $\ell$ used for forwarding the traffic is a system-wide parameter, which is determined by the security level desired in the system. For simplicity, and without losing generality, let $n$ be the number of users in the system. Accordingly, the anonymity is defined for two parties; the sender and the receiver of traffic. For the sender, the *anonymity set* is $n$, and the entropy of the probability distribution for a certain node being the sender is $H_s = \log_2(n)$. On the other hand, the anonymity set provided for the receiver is determined by the probability distribution achieved after the fixed number of hops used in the system. Let the distribution of the final node selected in a *random walk* after $\ell$ hops be $\pi^\ell$, where $\pi^\ell = [\pi_i^\ell]^{1\times n}$, then the anonymity of the receiver of the traffic (the last hop in the walk) is $H_r$, which is given as:

$$H_r = -\sum_{i=1}^{n} \pi_i^\ell \log_2 \pi_i^\ell \tag{7.1}$$

Using ((7.1)), we define the *anonymity set* $A^\ell$ as

$$A^\ell = 2^{H_r} \tag{7.2}$$

Given a random walk on a graph with certain properties—see section 7.1.2 for details—that random walk has a unique bounding or stationary distribution (defined in ((7.4))) which captures the maximum achieved entropy.

The idea of using social networks to provide anonymity and empower privacy is appealing for several reasons. First, social networks have been known for certain algorithmic properties that make the maximal entropy of a random walk achievable within a few hops from any node in the social graph [**?**]. On the other hand, unlike anonymity on fully structured graphs, such as that provided by Tor [**?**]—in which recruiting relays and maintaining trusted ones is a challenge [**?**]—social network-based anonymity systems could be a alternative with rich trust characteristics. This trust is the main ingredient used for reasoning about the potential of these networks for safeguarding users privacy [**?**]. Indeed, social network-based anonymity systems, such as MCON [**?**] are

known for their desirable characteristics, which are challenging in traditional mixing-based anonymous communication systems [**?**, **?**, **?**]. In the following, we further formalize the system settings as a graph-theoretic problem.

### 7.1.2 Formalization (for static graphs)

Let $G = (V, E)$ be an undirected and unweighted graph where $|V| = n$, $|E| = m$, $V = \{v_1, v_2, \ldots, v_n\}$, and $e_{ij} \in E$ iff $v_i \sim v_j \in V$. We define $\mathbf{A} = [a_{ij}]^{n \times n}$ as the adjacency matrix of $G$ where $a_{ij} = 1$ iff $e_{ij} \in E$ or 0 otherwise. Define the Markov chain on the graph $G$ following the transition matrix $\mathbf{P}$ which is defined according to $\mathbf{P} = [p_{ij}]^{n \times n}$ where:

$$p_{ij} = \begin{cases} \frac{1}{\deg(v_i)}, & v_i \sim v_j \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

A unique stationary distribution is defined for the Markov chain over the transition probabilities defined above if the Markov chain is ergodic—requiring it to be both *irreducible* and *aperiodic* [**?**]. Theorem 4 states such distribution.

**Theorem 4** *(Stationary distribution on static graph) For an undirected and unweighted graph $G$, the stationary distribution of the Markov chain defined over $G$ according to transitions in ((7.3)) is the probability vector, given as $\pi = [\pi_i]^{1 \times n}$, where*

$$\pi_i = \deg(v_i)/2m \tag{7.4}$$

**Proof 1** *The proof is a special case of the weighted graph case discussed in section 7.2 and follows from Theorem 5*

Using the model in ((7.1)) and the distribution in ((7.4)), we define the maximal (in size) anonymity set following the same model as in ((7.2)) as $A^{\infty} = 2^{H_r^{\infty}}$, where:

$$H_r^{\infty} = -\sum_{i=1}^{n} \left(\frac{\deg(v_i)}{2m}\right) \log_2 \left(\frac{\deg(v_i)}{2m}\right) \tag{7.5}$$

### 7.1.3 Lower-bound on the Achieved Receiver Anonymity

In [**?**], Nagaraja considered defined the average distribution achieved after $\ell$ hops from any potential source in the social graph as the anonymity achieved of every potential

source. While this captures the average performance in the system, it simply does not show the worst case scenario observed at the lower-bound of the achieved anonymity for receivers. Here, we revise Nagaraja's definition in [**?**] and outline a straightforward fix for the measure of the anonymity provided in a system that uses walks on the social graph.

Without losing generality, let $\ell$ be a system-wide parameter, which represents the number of hops from the source to the destination (or receiver) in the graph, and each node between them is chosen uniformly at random from its predecessor. For each source $v_j$ (for $1 \leq j \leq n$), we define the probability distribution after $\ell$ hops as $\pi^{\ell}(v_j) = [\pi_i^{\ell}(v_j)]^{1\times n}$ for (for $1 \leq i \leq n$). The anonymity achieved in the system is bounded below by the entropy achieved in the probability distribution obtained by walking from the worst source in the graph:

$$H_r \geq \inf_{v_j}\{-\sum_{i=1}^{n} \pi_i^{\ell}(v_j)\log_2 \pi_i^{\ell}(v_j)\} \tag{7.6}$$

By extending ((7.2)) to the case in ((7.6)), we get the following

$$A^{\ell} = 2^{H_r} \geq 2^{\inf_{v_j}\{-\sum_{i=1}^{n} \pi_i^{\ell}(v_j)\log_2 \pi_i^{\ell}(v_j)\}} \tag{7.7}$$

The intuition of this lower bound is very simple, and follows from the definition. Technically, this lower bound follows the classical theoretical trend in security: proving lower bounds of security (or anonymity as it is the case in hand) would enable us to guarantee, in the worst time, that our system would perform better than this bound for every user. On the other hand, considering the average case for achieved entropy might be very deceiving since many receivers are likely not to achieve this average bound.

## 7.2 Dynamic Graphs for Anonymity

Here, we extend the findings in the literature on using static graphs as mixers for anonymous communication to the case of the dynamic graphs. Such dynamic graphs arise naturally in many contexts due to social churn imposed by node and edge dynamics (joining and leaving social networks). It is worth noting that this is the first work of its own type to consider extending such results for building anonymous communication systems on top of dynamic social graphs.

### 7.2.1 Formalization: The case of Dynamic Graphs)

The dynamic graph is a simple generalization of the static graph used in literature. In particular, $G = \{G^{(i)}\}$ for $1 \leq i \leq t$ is a dynamic graph over $t$ time periods. Let $G^{(i)} = (V^{(i)}, E^{(i)})$ for $1 \leq i \leq t$, where $|V^{(i)}| = n^{(i)}$ and $|E^{(i)}| = m^{(i)}$, be an unweighted and undirected graph (later we extend that to the weighted graph case). Let $V^{(i)} = \{v_1^{(i)}, v_2^{(i)}, \ldots, v_{n^{(i)}}^{(i)}\}$ and $E^{(i)}$ be the set of pairs of vertices $v_j^{(i)}$-$v_k^{(i)}$ if both nodes $v_j^{(i)}$ and $v_k^{(i)}$ in $V^{(i)}$ are connected to each other. For $G^{(i)}$, we define $\mathbf{A}^{(i)}$ where $\mathbf{A}^{(i)} = [a_{jk}^{(i)}]^{n^{(i)} \times n^{(i)}}$ (the superscription is used as part of the notation, and does not mean power), where:

$$a_{jk}^{(i)} = \begin{cases} 1 & v_j^{(i)} \sim v_k^{(i)} \in G^{(i)} \\ 0 & \text{otherwise} \end{cases}. \tag{7.8}$$

For the same graph $G^{(i)}$, we define the transition probability matrix $\mathbf{P}^{(i)}$ such that $\mathbf{P}^{(i)} = [p_{jk}^{(i)}]^{n^{(i)} \times n^{(i)}}$, where:

$$p_{jk}^{(i)} = \begin{cases} 1/\deg\left(a_{jk}^{(i)}\right) & v_j^{(i)} \sim v_k^{(i)} \in G^{(i)} \\ 0 & \text{otherwise} \end{cases}. \tag{7.9}$$

Extending and generalizing ((7.8)) and ((7.9)) to the weighted case is easy if weights are given on edges in the graph. We define

$$a_{jk}^{(i)} = \begin{cases} w(v(i)_j, v(i)_k) & v_j^{(i)} \sim v_k^{(i)} \in G^{(i)} \\ 0 & \text{otherwise} \end{cases}, \tag{7.10}$$

where $w : E^{(i)} \to \mathbb{R}$ is a weight function that assigns real-valued weights to edges in $G^{(i)}$. Using ((7.10)), we define the degree of a node to be in terms of weights associated with edges for which that node is an end-vertex, as

$$\deg^w\left(v_j^{(i)}\right) = \sum_k w(v_j^{(i)}, v_k^{(i)}), \quad v_j^{(i)} \sim v_k^{(i)} \in G^{(i)}. \tag{7.11}$$

Notice that ((7.11)) can also be written as $\deg^w\left(v_j^{(i)}\right) = \sum_k a_{jk}^{(i)}$—where $a_{jk}^{(i)}$ is defined in ((7.10)). Using ((7.11)), we can compute $\mathbf{P}^{(i)} = [p_{jk}^{(i)}]$ for weighted graphs, where

$$p_{jk}^{(i)} = \begin{cases} w(v_j^{(i)}, v_k^{(i)})/\deg^w\left(v_j^{(i)}\right) & v_k^{(i)} \sim v_k^{(i)} \in G^{(i)} \\ 0 & \text{otherwise} \end{cases}. \tag{7.12}$$

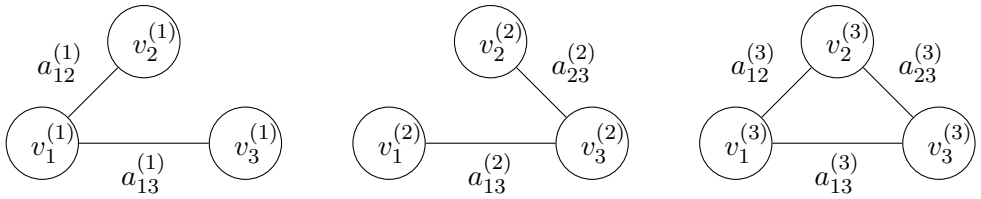



Figure 7.1: Simple example of dynamic graph. $a_{12}^{(i)} = w(v_1^{(i)}, v_2^{(i)})$ is as per the definition. One or more of the weights $a_{12}^{(i)}$ could be zero.

In a matrix form, $\mathbf{P}^{(i)}$ can be defined as $\mathbf{P}^{(i)} = (\mathbf{D}^{(i)})^{-1}\mathbf{A}^{(i)}$ where $\mathbf{D}^{(i)}$ is a diagonal matrix computed from $\mathbf{A}^{(i)}$, where the diagonal element $d_{jj}^{(i)}$ in $\mathbf{D}^{(i)}$ is the sum of ones in the $j$-th row in $\mathbf{A}^{(i)}$ (that is, the degree of node $v_{ix}$ in $G^{(i)}$). At any time slot $i$, we define the bounding distribution of the Markov chain on the graph $G_i$ as in literature defined $[\deg(v_j^{(i)})/2m^{(i)}]$. It is, however, unclear how to proceed with the different snapshots of the same graphs at different times.

For example, as shown in Figure 7.1, both nodes $v_1^{(1)}$ and $v_2^{(1)}$ are connected, but not with their future images—$v_1^{(2)}$ or $v_1^{(2)}$ and $v_2^{(2)}$ or $v_2^{(3)}$, respectively. This also applies to states in the future not connected to the past images. In the following, we investigate several techniques for modeling the dynamic social graph as a graph where transitions from future states to past states is possible. Techniques utilize here are generic, and can be used to any graph with multiple labels.

Prior work in the literature has tried to model dynamic graphs as *3-mode tensor* [**?**] or *union multigraph* [**?**]. However, while the first uses high dimensionality—making computations on the tensor computationally expensive, the second technique reduces dimensionality and loses some information about the graph. Indeed, the second technique computes the union between multiple graphs (edge- and node-wise) and omits any potential multiple edges between two nodes in the union. While this is meaningful to understand a union snapshot of multiple graphs, demonstrate connectivity characteristics of the union graph driven from multiple attributes, and potentially other benefits, it does not capture the "depth" of edges and does not differentiate between different edges based on their "real value". For example, while edges in the union multigraph are all the same, some in reality might be the result of multiple edges whereas others

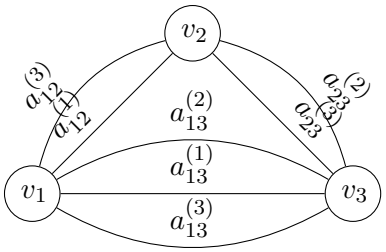



Figure 7.2: Simple example of converting a dynamic graph (in figure 7.1) into multigraph by collapsing all images of a node to the node itself (the resulting nodes of the graph are the result of set union) and creating multiple-edges (corresponding to multiset union of the individual graphs).

could be the result of a single edge.

### 7.2.2 Dynamic graph as a multigraph

As per the formalism in section 7.2.1, a dynamic graph over $t$ time slots can be viewed as a multigraph: all nodes that correspond to a certain past state are collapsed under the same label in the present state. Accordingly, if two nodes that have been labeled with two different labels in the transformation process from multiple graphs to multigraph had an edge between them, the edge is created in the multigraph. Since the multigraph includes states of nodes in the past, current, and future time (for the current snapshot of the graph), multiple edges are potentially created between two labels. Such edges could be weighted or unweighted, depending on the original multiple snapshots representing the dynamic graph.

Formally, for the dynamic graph $G = \{G^{(i)}\}$ described in section 7.2.1, we define a multigraph $\mathbb{G}$ as $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where

$$\mathbb{V} = \bigcup_{i=1\ldots t} \{V^{(i)}\}, \text{ and } \mathbb{E} = \biguplus_{i=1\ldots t} \{E^{(i)}\}. \tag{7.13}$$

Notice that $\cup$ is a *set union*, which does not allow repetition of vertices, whereas $\uplus$ is a *multiset union*, which allows edge repetition. When $\mathbb{E}$ is computed, the index that corresponds to the time of the edges in $E_i$ can be removed for simplicity. A simple toy

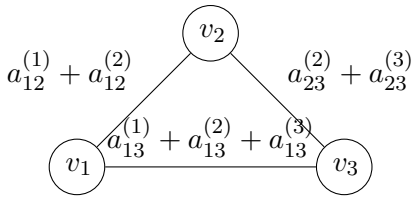



Figure 7.3: A simple example of multigraph (shown in figure 7.2) conversion into weighted graph by summing weights of edges between every pair of nodes.

example of transforming the multiple snapshots of the dynamic graph in Figure 7.1 into a multigraph is in Figure 7.2.

Our formalization above of the graph as a multigraph (rather than union multigraph as per the way defined in [**?**]) follows the intuition of what a dynamic graph could yield of associations at any time[1] . At a time $i$, where $1 \leq i \leq t$, constructing the proper graph for operating a potential system, like mixing-based anonymous communication system, and maintaining the same information driven from the original multiple snapshots of the graph would be possible. Furthermore, assigning different weights to different associations, based on the history of the association, would be also possible (see below for details).

The literature of graph-theory, however, provides us some powerful tools to reduce the dimensionality of the multigraph above. Indeed, the adjacency matrix representation of multigraph accepted in the literature [**?**] considers entries in this matrix as the number of edges (or sum of weights) between nodes. Accordingly, reducing the multigraph into a "weighted" graph representation is straightforward.

In the following section, we elaborate on this notation and show a transformation of the multigraph into a weighted graph.

### 7.2.3 Dynamic graphs as weighted-graphs

Now, we convert the dynamic graph model represented as a multigraph, as in ((7.13)), into a weighted graph. We generalize formalizations in section 7.2.1. In particular, the

[1] The weights used to construct the dynamic graph for anonymous communication are results of associations up to the current time. A node does not need to know the future links with other nodes, by the past ones.

model in ((7.10)) can be rewritten (for weighted undirected graph) as $\mathbf{A} = [a_{jk}]^{n \times n}$—here, $n = |\mathbb{V}|$—where

$$a_{jk} = \sum_{i=1...t} w(v_j^{(i)}, v_k^{(i)}), \;\; v_j^{(i)} \sim v_k^{(i)} \in G^{(i)} \forall i. \tag{7.14}$$

Similarly, we extend the model in ((7.11)) into

$$\deg^w(v_j) = \sum_{i=1...t} \deg^w(v_j^{(i)}) = \sum_{\forall k} a_k^{(j)} \tag{7.15}$$

$$= \sum_{\forall k} \sum_{i=1...t} w(v_j^{(i)}, v_k^{(i)}), v_j^{(i)} \sim v_k^{(i)} \in G^{(i)}. \tag{7.16}$$

We can further extend the transition probability formulation to cover the weighted graph by plugging both ((7.14)) and ((7.15)) into a similar model to that of ((7.12)), to get $\mathbf{P} = [p_{jk}]^{n \times n}$, where

$$p_{jk} = a_{jk} / \deg^w(v_j) \tag{7.17}$$

For a random walk defined on $\mathbb{G}$ according to the transition probability defined in ((7.17)), the following theorem states the stationary distribution. This theorem (and the proof herein) are essential for latter results on characterizing and operating on dynamic graphs. Also, the proof of Theorem 4 follows similarly as in the proof below.

**Theorem 5** *Let* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *be a* connected, undirected, *and* weighted *graph defined as in (*(7.13)*). For a random walk following transition probabilities as in (*(7.17)*), the stationary distribution is defined as* $\pi = [\pi_i]^{1 \times n}$ *(for* $n = |\mathbb{V}|$*), where:*

$$\pi_i = \deg^w(v_i) / \sum_{k=1...n} \deg^w(v_k) \tag{7.18}$$

**Proof 2** *To prove Theorem 2, we show (1) that* $\sum_{i=1...n} \pi_i = 1$*, and (2)* $\pi = \pi \mathbf{P}$*, where* $\mathbf{P}$ *and* $\pi$ *are defined in (*(7.17)*) and (*(7.18)*).*

*To show that the first condition holds for the given distribution, we simply sum* $\pi_i$ *in (*(7.18)*) over all* $i$ *to get:*

$$\sum_{i=1...n} \pi_i = \sum_{i=1...n} ([\deg^w(v_i)] / [\sum_{j=1...n} \deg^w(v_j)])$$

$$= [\sum_{i=1...n} \deg^w(v_i)] / [\sum_{j=1...n} \deg^w(v_j)] = 1 \tag{7.19}$$

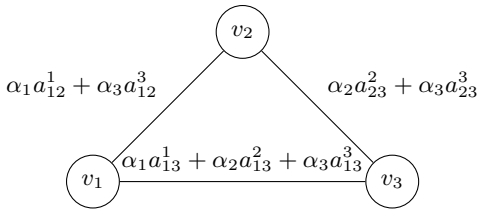



Figure 7.4: A toy example of the generalized weighted graph model to express dynamic graphs—the same graph in Figure 7.3 is used.

*To show that the second condition holds, let* $\Delta = \pi\mathbf{P}$*, where* $\Delta = [\delta_1, \delta_2, \ldots, \delta_n]$*. Let us look at the structure of* $\delta_k$ *for any* $k$*. Using* $\pi$ *and* $\mathbf{P}$ *defined above, we get* $\delta_k = \pi\mathbf{p}^k$ *(where* $\mathbf{p}^k$ *is the* $k$*-th column in the matrix* $\mathbf{P}$ *defined in ((7.17)*)*).*

$$\begin{aligned}\delta_k = \pi\mathbf{p}_k &= \sum_{\forall i}(\deg^w(v_i)/\sum_{j=1\ldots n}\deg^w(v_j))(a_{ik}/\deg^w(v_i)) \\ &= \deg^w(v_k)/\sum_{j=1\ldots n}\deg^w(v_j) = \pi_k \end{aligned} \tag{7.20}$$

*By applying the result in ((7.20)) for all* $k$*, we obtain* $\Delta = \pi$ *thus* $\pi = \pi\mathbf{P}$*, which concludes the proof.*

### 7.2.4 Generalized weighted graphs

In many natural social contexts, recent associations are more valued than older ones, or vice-versa. Accordingly, a general framework for quantifying the potential of any system on top of social networks should consider implicit social network characteristics, such as link age, in addition to the explicit differences among links captured by the topological structure. We generalize the model in section 7.2.3 to accommodate for implicit values of associations over time. Without losing generality, let $\alpha_i$ (for $1 \leq i \leq t$) be a set of parameters that take numerical values. An extension of the social graph model in ((7.14)) is as follows:

$$a_{jk} = \sum_{i=1\ldots t}\alpha_i w(v_j^{(i)}, v_k^{(i)}),\ v_j^{(i)} \sim v_k^{(i)} \in G^{(i)} \forall i. \tag{7.21}$$

The rest of the model in section 7.2.3, particularly in ((7.15)) through ((7.20)), holds for this generalization after adjusting $a_{jk}$ as in ((7.21)). A toy example demonstrating the adjustment of weights in Figure 7.3 is shown in Figure 7.4.

## 7.3 The Datasets

Here we demonstrate the results of our modeling of dynamic social networks for anonymous communication systems considering several strategies for obtain the underlying social graphs. Before exploring results, we first discuss the datasets and the method used for extracting these structures—including the dynamic social structures—from them.

Our sources of data are two datasets which represent interaction graphs. These datasets are for Facebook [**?**], an online social network, and DBLP [**?**], a co-authorship network. More details are in the following.

### 7.3.1 The DBLP Dataset and its Preprocessing

The DBLP datasets represents co-authorship graph, where nodes are authors and a link between two authors implies that the authors have co-authored a paper. The original DBLP dataset consists of $943,316$ nodes and $6,379,554$ edges between them, for publication records until May 2011 in computer science areas. The graph consists of $40,685$ disconnected components and all edges are between $892,565$ nodes. After removing minor disconnected components, the remaining graph consists of $769,642$ and $3,051,127$ undirected edges. To generate dynamic graphs from the largest connected component of the DBLP graph, we limit ourselves to the period of 2006 to 2010 inclusive. We select each author who has publications at each and every of these years. The result is a multigraph where two nodes would have an edge if they co-authored a paper in a given year—and number of such papers per year is used as weight for weighted graphs. Multiple edges could be created between two authors if they co-authored over multiple years. Multiple edges are labeled with respect to the year of publication. The final multigraph has $46,994$ nodes and $458,736$ edges. We decompose each multigraph to multiple-graphs with respect to the edge label. Finally, as some nodes who published in the given period could be isolated in a certain year, we remove these nodes so as each resulting graph is connected—such nodes will be prohibited from using the system. Statistics of the different resulting graphs are shown in Table 7.1. Degree distribution of the same set of graph is shown in Figure 7.5. For our study, we consider several cases of the same graph including both weighted and unweighted, with respect to the time. All variations of the used graphs in our experiments are as follows:

- Unweighted graph with respect to each year (5 graphs). Each graph is obtained by omitting the weights (number of publications) on the edges.

- Unweighted single graph representing the entire dataset (1 graph). This graph is computed as the node and edge set union over the 5 different snapshots each of which correspond one interval of dynamics (in step 1).

- Weighted single graph representing the entire dataset. The weight on an edge connecting two nodes is the sum of all weights of edges between these nodes over time from the beginning to the end of recording the graph structure. This is, weights edges between two nodes correspond to the *total* number of publications between each two authors representing to these nodes in all times.

- Weighted multiple snapshot graphs (up to each year; 5). A graph, say $G_{1i}$ combines all nodes in $G_1$ to $G_i$ and the edges between them. Because it is weighted, the weight on an edge between two nodes in $G_{1i}$ is the sum of all edges in the union of the graphs that constitute $G_{1i}$ and which are between these two nodes.

- Unweighted multiple snapshot (up to each year; 5). These graphs are obtained using the same method as in the previous step but without weights on edges.

- Weighted graphs with weights assigned based on the age of the link. We use geometrical ($2^{1-x}$ where the newest graph has $x = 1$ and the oldest has $x = 5$) and reciprocal ($1/x$) decay distributions (2 graphs).

In Table 7.1 and Table 7.2, the basic structural statistics and properties shown as follows

- *Graph size:* number of nodes and number of edges.

- *Clustering coefficient:* is the average (thus in [0, 1]) of local clustering coefficient for all nodes. The local clustering coefficient for a node is the fraction of possible triangles that go through that node.

- *Diameter:* the longest of eccentricities among all nodes in the graph. The eccentricity of a node is the longest shortest path from it to other nodes in the graph.

Table 7.1: Statistics of DBLP time-varying graphs. Metrics of comparison are number of nodes ($n$), number of edges ($m$), average clustering coefficient, diameter, and radius.

| | # nodes | # edges | clustering | diameter | radius |
|---|---|---|---|---|---|
| DBLP (1) | 31704 | 71994 | 0.483 | 26 | 14 |
| DBLP (2) | 33012 | 79475 | 0.480 | 27 | 14 |
| DBLP (3) | 33923 | 84125 | 0.467 | 24 | 13 |
| DBLP (4) | 33071 | 82282 | 0.453 | 23 | 12 |
| DBLP (5) | 26150 | 62161 | 0.419 | 24 | 13 |

- *Radius:* shortest eccentricity.

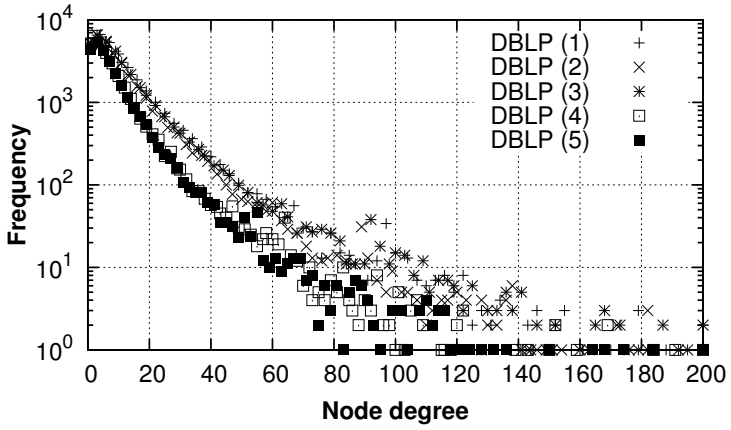



Figure 7.5: Degree distribution of the different snapshots of the DBLP dataset. Notice slight difference in the degree distribution across graphs.

### 7.3.2 The Facebook Dataset and its Preprocessing

The Facebook dataset [**?**] is for wall posts in New Orleans regional network, and spans the period from 2004 to 2009. A link between two nodes indicates that the first node has interacted with the second node. Further details on statistics of the entire dataset is in [**?**]. The volume of interaction begins slow and increases as the time goes. To obtain a dynamic graph from this dataset, we limit ourselves to the last 30 months of

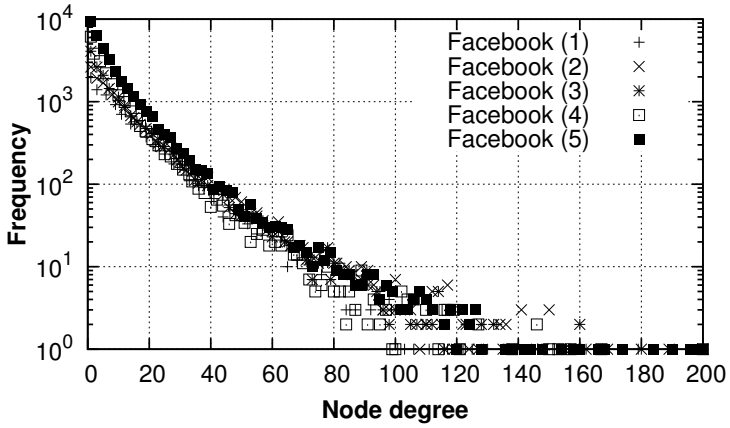



Figure 7.6: Degree distribution of the different snapshots of the Facebook dataset. Notice no major change in the degree distribution across graphs.

Table 7.2: Statistics of Facebook time-varying graphs. Metrics of comparison are number of nodes ($n$), number of edges ($m$), average clustering coefficient, diameter, and radius.

| | # nodes | # edges | clustering | dimeter | radius |
|---|---|---|---|---|---|
| Facebook (1) | 9154 | 23245 | 0.102 | 19 | 10 |
| Facebook (2) | 13288 | 37908 | 0.101 | 18 | 10 |
| Facebook (3) | 16540 | 42427 | 0.092 | 19 | 10 |
| Facebook (4) | 23879 | 59190 | 0.085 | 21 | 11 |
| Facebook (5) | 35665 | 86525 | 0.084 | 18 | 10 |

interactions. Because the dataset itself is small and does not guarantee that interactions between the same set of nodes persist over the period of time of interest, we consider all interactions in the period of the 30 months and consider each graph snapshot over a period of 6 months. The resulting five graphs are shown in Table 7.2 and their degree distributions are shown in Figure 7.6. Notice the high variability in graph size, which is explained by the growth of Facebook in that period [**?**]. The same variations used to generate the test datasets for DBLP above are also used here. We omit details for the lack of space.

## 7.4 Results

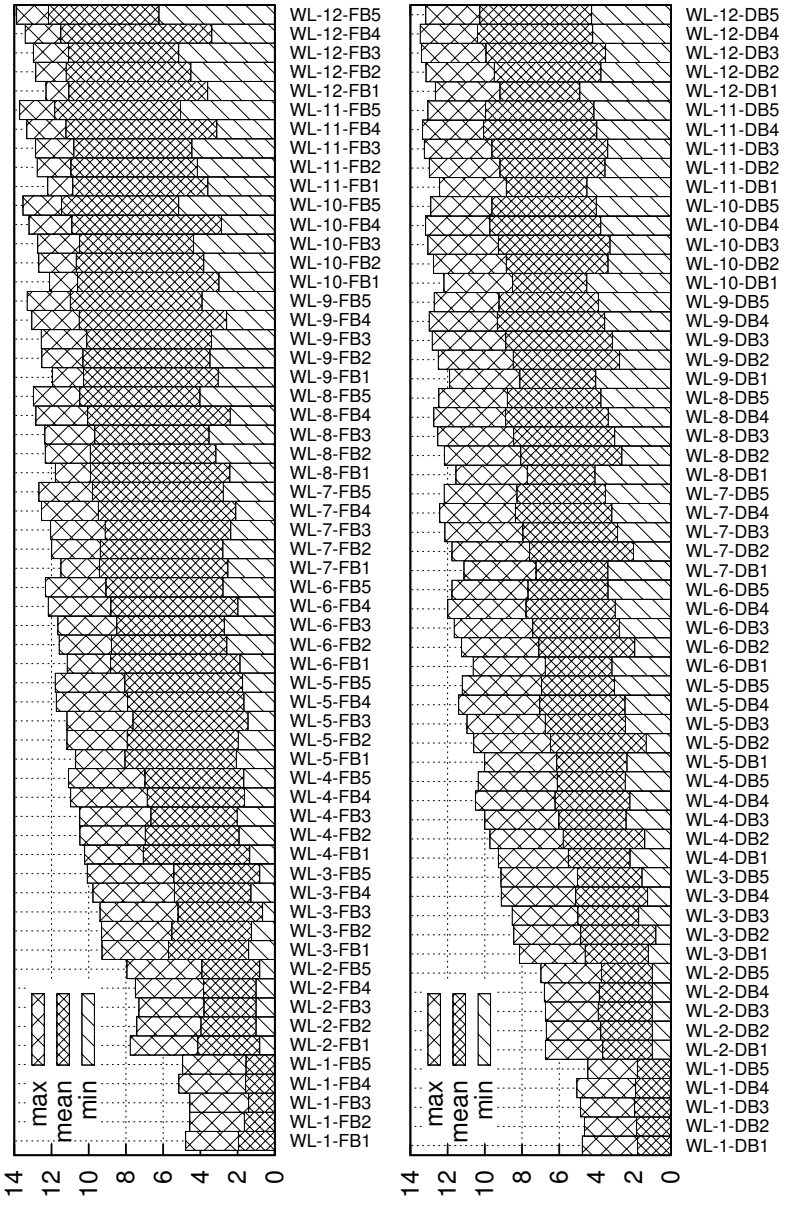
Here we outline the results of utilizing the different social graphs obtained in section 7.3. Our main measurement metric is the achieved anonymity in terms of the total entropy in the distribution of the last hop in a random walk, as the length of the random walk increases (computed as in ((7.1)). We keep in mind that potential utilization of social graphs for anonymity systems would be subject to the performance of this systems, which necessitate to a short random walk length. We consider walk lengths varying from 1 to 20 steps, where walk lengths of $1 - 12$ are demonstrated in most experiments. As the entropy varies depending on the source of the random walk, we are interested in the maximum, minimum (advocated in section 7.1.3 and expressed in ((7.6))), and mean entropies for a given dataset as the walk length increases.

### 7.4.1 Original unweighted graphs

We first consider operating the anonymity system on top of the graphs shown in Table 7.1 and Table 7.2. The results are shown in Figure 7.9. Two observations are made on these figures. First, the lower-bound on the achieved entropy or the entire system, according to the model in ((7.6)), is much smaller than that of the average and maximum entropy, for any walk length. This in particular tells that the measure of the lower-bound on the entropy, while theoretically appealing for the guarantees advocated in section 7.1.3, they do provide a representative measure for the whole set of nodes in the system or graph. Second, we observe that both the mean and the maximum of the entropy in each of the graphs stabilizes, and reaches its potential maximum entropy within a relatively smaller number of steps, corresponding to shorter random walk length. This indeed interesting, and agrees with prior work in [**?**], despite that the results in the prior work have been on relatively a faster mixing social graph [**?**].

### 7.4.2 Dynamics as Weights

We consider our main thesis in this chapter of modeling dynamics of social graphs as weights on edges. We use the method in 7.2.3 for generating these graph with weights. The method of obtaining the graphs is explained in 7.3. The results, for both DBLP and Facebook datasets are shown in Figure 7.8(a) and Figure 7.8(b). Similar observations




(a) Facebook (b) DBLP

Figure 7.7: Min-mean-max bar diagram of the achieved anonymity by utilizing the individual datasets of the different social graphs (snapshots) for several random walk lengths. DBLP (x) is abbreviated into DBx for space constraints.

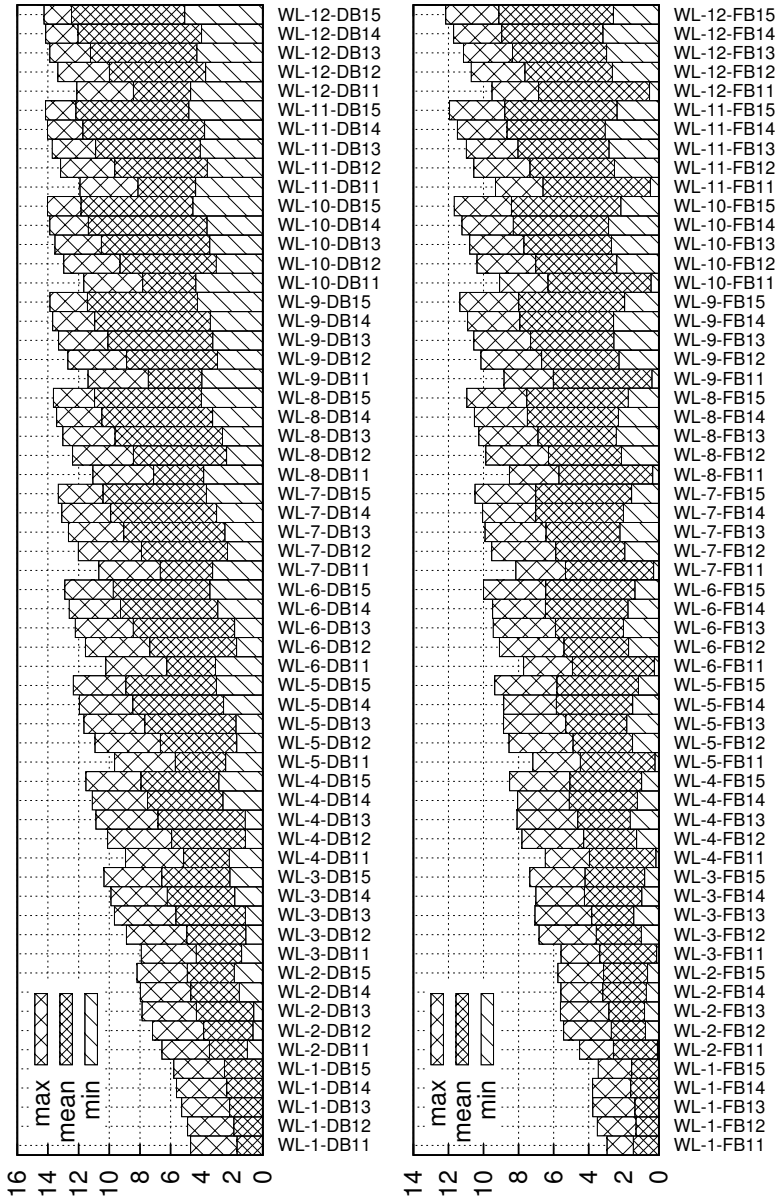
on the results for the tendency of weights is made as on the previous measurements. Furthermore, the *general* tendency of improvement of the entropy value as the time goes is made clear in both figures.

### 7.4.3 Unweighted Dynamic Graphs

We consider removing weights from the different dynamic graphs to observe how this affects the entropy as walks length increases. For the same experiment above, with the only difference being deletion of weights, we get the results in Figure 7.9(a) and Figure 7.9(b) for DBLP and Facebook, respectively. The most important point made clear in this experiment is that unweighted graph generated from these weighted dynamic graphs provide higher entropy for the same walk length (over the same number of nodes).

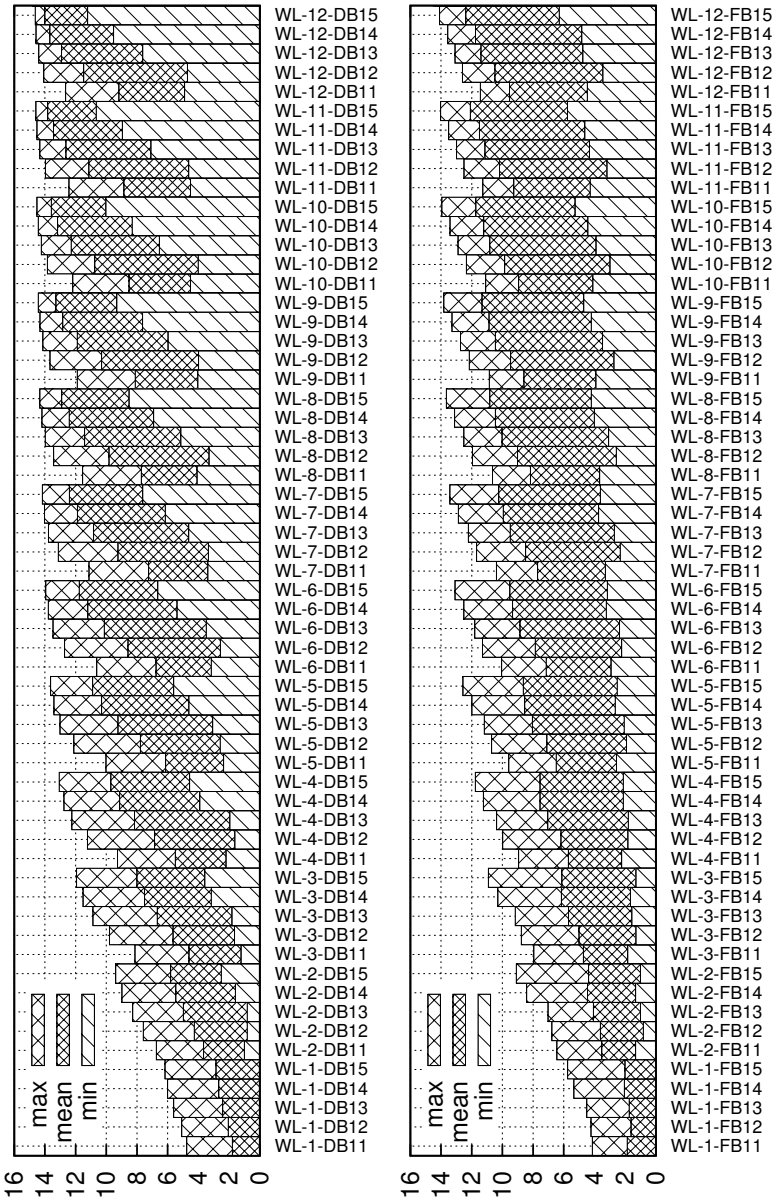
### 7.4.4 Dynamics as Differential Weights

We consider the potential ways of assigning the different weights on the social graph, based on the age of the link, and how this impacts the achieved anonymity on these graphs. We consider the graphs constructed from the multiple-snapshots, according to the way described in section 7.2.3 and section 7.2.4. We use the result in section 7.2.3 to generate a "linear" weighting factor (the coefficient is 1), and thus the weight of an edge is the number of interactions between the two nodes over all years. We consider the same graph generated in this step without weights as well. Finally, we use the model in section 7.2.4 to generate generalized weighted graphs, where weights are formed according to the reciprocal or geometrical decay distributions explained in 7.3. The results of the measurements are shown in Figure 7.10(a) and Figure 7.10. In these measurements, and somewhat counter-intuitively, we observe that the unweighted graph model results in best entropy (in all three categories: min, mean, and max)—Further details are in the discussion section. On the other hand, we also observe that in all of these graphs, the achieved anonymity is good enough (as a portion of the maximal) even with a walk length of 10, suggesting the usefulness of this design. Both remarks apply to both datasets, though bias is a lot higher in Facebook than in DBLP.



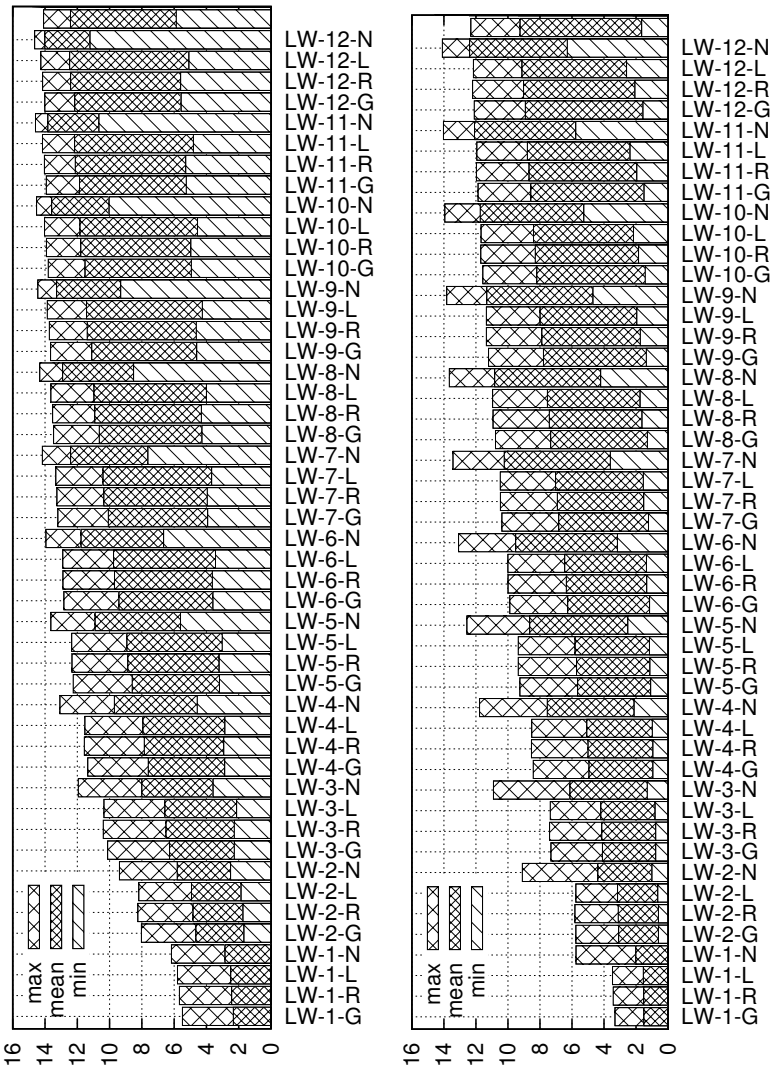

(a) DBLP (b) Facebook

Figure 7.8: The anonymity achieved (as entropy) by walking on the different *weighted* graphs in DBLP and Facebook constructed according to the dynamic method construction described earlier. Weights are linear sum of all original edge weights, and 1x indicates that the graph is constructed by computing the union graph on $G_1, \ldots, G_x$.




(a) DBLP (b) Facebook

Figure 7.9: The anonymity achieved (as entropy, represented on the x-axis) by walking on the different *unweighted* graphs in DBLP and Facebook constructed according to the dynamic method construction described earlier. No weights are used and 1x indicates that the graph is constructed by computing the union graph on $G_1, \ldots, G_x$.




(a) DBLP (b) Facebook

Figure 7.10: Min-mean-max bar diagram of the achieved anonymity (the entropy reflected on the x-axis) by utilizing the generalized wieghted graph model to capture dynamics; G stands for geometrical distribution, R for reciprocal, L for linear and N for no weights.

### 7.4.5 Comparative Scenario

We consider all scenarios mentioned above, for each of the datasets we had, and for a fixed random walk length to compare them relatively and draw final conclusions on the impact of the underlying social structure on the achieved anonymity. We consider the random walk length $\ell = 10$, and experiment for both datasets to compute the entropy—both mean and max. The results are shown in Figure 7.11 through Figure 7.12.

## 7.5 Analysis and Discussion

In most of the measurements of the entropy on the distribution of the random walk after $\ell$ hops, we observe a relatively good entropy which supports the claimed efficiency advocated in this work and [**?**]. However, this entropy, for example is not as high in some graphs, especially those the single snapshots that consider the graph at one time period, and those resulting from assigning weights on the graphs corresponding to the richness of the edges. This pattern is shown in both datasets, which call for explanations.

One potential explanation of the relative difference between the achieved entropy in the individual graphs and that obtained from the graphs computed using our dynamic graph model is the inherit increment in the size of the resulting final graph. For example, while the largest DBLP graph is about 36,000 nodes, the final graph of the DBLP after using our model that considers graph dynamics would result into about 44,000 nodes in the largest connected components. This and the fact that the graph becomes richer of more edges that connect multiple components in the graph improves the mixing characteristics of the graph, which ultimately improves the achieved entropy as $\ell$ increases.

On the other hand, this by itself does not explain the difference between the achieved entropy in both weighted and unweighted graphs, even for the graph with the same number of nodes and edges. For example, when $\ell = 10$, the achieved entropy on DBLP-15 when weighted is 14 while it's 14.5 for the same graph when it is unweighted—while the difference in entropy is small, i.e., 0.5, the 14.5 bits of entropy provide about $23,170$ anonymity set whereas only $16,384$ are provided for the other case, which translates to more than $6,786$ of difference in anonymity set. One possible explanation of this behavior is the intuitive meaning of weighting graphs: by assigning weights on edges,

we are biasing the random walk on such graphs and favoring a node over another of being reaching at any time when running the random walk. This is, some nodes will be more likely reached whereas other nodes while less likely reaching by the random walk which definitely decreases the potential set of nodes being used as a last hop in the random walk. This intuitive meaning explains the difference in the entropy in both cases.

Unexpectedly, both entropy and anonymity sets are decreased (and greatly) using the weighted graphs that model dynamic structure, which raises the question about the usefulness of using these graphs for anonymous communication rather than using the unweighted version of them which only considers plain set union of nodes and edges by omitting weights. One potential explanation for this issue is that these weights are obtained by favoring some edges over others, which is more meaningful from an anonymity point of view, whereas edges in the unweighted graph simply make all relationships over time equal. In a realistic scenario, where potential insider attacker could exist to penetrate the system to get communicated messages the social overlay, the model, which considers links to be equal independent of their history or time of creation, could be problematic. Given this intuitive explanation of weights associated with edges, one would anticipate the use of weighted graphs in real-world scenarios despite this degradation in the achieved entropy and anonymity set sizes given their potential for minimizing harms due to edge infiltration.

## 7.6 Related Work

Exploiting static social networks for anonymous communication has been explored in [**?**], which has been discussed earlier and did not consider the dynamic graph case. Modeling of dynamic social graph and extracting a definition for the mixing time is done in [**?**], whereas sampling multigraph defined as node set union graphs is done in [**?**]. To the best of our knowledge, no prior work considered dynamic graphs in the context of the problem in hand and as per our method. A similar way of modeling dynamic graphs as snapshot of social graphs per fixed period of time (e.g., day) is used in [**?**]. The main finding in [**?**] is that topological properties for graph snapshots over time do not change greatly. We make such properties change by adding past associations, not limiting

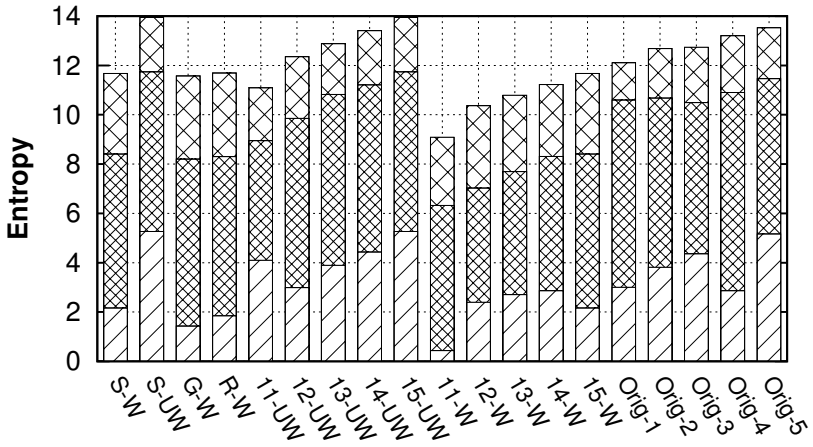



Figure 7.11: Average entropy for walk length of 10; all Facebook graphs (S. is single, G and R are geometrical reciprocal distributions of weights, W is weighted, and UW is unweighted, and numbers are to indicate which graph is used: 1-5 are original graphs whereas 11-15 are the dynamic graph model).

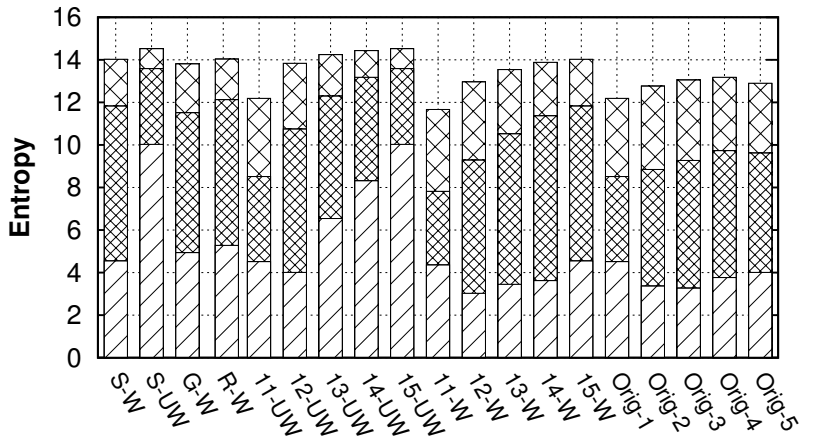



Figure 7.12: Average entropy for walk length of 10; all DBLP graphs (S. is single, G and R are geometrical reciprocal distributions of weights, W is weighted, and UW is unweighted, and numbers are to indicate which graph is used: 1-5 are original graphs whereas 11-15 are the dynamic graph model).

ourselves to the current associations, but rather with decaying weights.

There has been a large body of work in the literature on utilizing social networks for trustworthy computing, where social networks are used to bootstrap communication and distributed computing systems in order to improve their security and reliability. For example, and perhaps the closest application which uses similar algorithmic property, social network-based Sybil defenses—which look at preventing nodes in distributing computing settings from claiming multiple identities by tying digital to social identities—is a well-explored vein of research that produced many designs [**?**, **?**, **?**, **?**, **?**, **?**]. The main assumptions being assumed in these designs are fast mixing and trust-possessing social graphs, which have been challenged in [**?**] and [**?**], respectively. To the best of our knowledge, none of these designs handles the social network dynamics. A worthwhile future direction is to investigate incremental deployment issues of these designs under dynamics.

Dynamic social graphs [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**] have been considered by analysis in prior literature where evolution has been demonstrated and patterns have been outlined. Most of these studies have considered mining known (and almost always) simple parameters of social graphs, but not the mixing time and patterns used for anonymity.

An interesting recent work in [**?**] shows that as the time goes social graphs tend to have higher density and slightly better quality of properties such as the average shortest path, clustering coefficient, and degree. In [**?**], it is shown that as the time goes smaller communities tend to merge quickly into larger communities, which dominate about 60% of the graph size, making the "small-world" graph even smaller.

## 7.7 Summary

In this work we considered the problem of building anonymous communication systems on (unstructured) dynamic social graphs. We have pointed out an interesting relationship between dynamic structures and weighted graphs, and formalized the anonymity achieved under dynamics as a random walk on weighted graphs. We formulated the problem in hand, and shown the bounding distribution, which captures the maximal achieved entropy of a random walk on an anonymous communication system, which uses these dynamic structures. Through experiments on real-world datasets, we have

shown the potential of these dynamic structures, and despite their numerical disadvantage over unweighted versions, we have pointed out their benefits for anonymity for that they capture more meaningful structure that represents stronger ties.

# Chapter 8

# Conclusion and Future Works

In this dissertation we studied social networks properties that are widely used and accepted for designing security primitives or for improving the operation of distributed computing systems. We have shown that several of such properties do not exist naturally in social networks with the quality assumed by prior work, and others do not exist to support the operation of such systems. We measure the mixing time—a major property used for building trustworthy systems on social networks, the expansion, and the betweenness in social graphs. Motivated by these measurements and findings, we consider look at how practices widely used in the literature affect the the quality of these properties in social networks. We examined at how omitting directionality of edges in social networks affect the mixing time, and studied the consequence this effect has on the operation of systems built on top of social networks. We found that such practice always overestimate the security guarantees provided by systems tested on altered graphs. Finally, we explored reasons behind the quality of the mixing time, and propose several heuristics to improve it.

Motivated by the lack of work on accounting for differential trust in social network-based applications—Sybil defenses in particular, we developed four designs and used them to account for trust in a benchmark technique of Sybil defense (SybilLimit). We empirically demonstrated that these designs improve the security of this defense at some reasonable cost. We propose to extend these algorithms to other applications, Sybil defenses as well as other routing and information dissemination applications on top of social networks and rely on trust and social graphs' structure in their operation.

this direction has several potential future work that go beyond this thesis, including the following:

- The potential of the designs to capture further properties of the social graphs in the context of applications built on top of these graphs. While these primitives affect the quality of the social networks properties (such as the mixing time), it is unclear how they affect other properties that can be also used for building applications on top of social networks. For example, on a graph that uses interactions rather than social connections, one would expect more clear community structure that corresponds to different betweenness, closeness, and clustering coefficient characteristics than that of the original social graph that considers social links only. This hypothesis is particularly supported by our preliminary findings that interaction graphs have a lower mixing than that of the original social graphs using social links. We would like to investigate this direction emphasizing on the properties previously measured and how these designs affect them. Further, we propose to investigate how the change in the quality of the properties affect the applications on top of the social graphs.

- In both of the parameterized design that we have suggested in our prior work (the originator-biased and the lazy), we have considered some assumptions to simplify the operation of the designs. For example, we considered that each design uses a globally fixed parameter ($\alpha$) that is used by each node in the graph. While this simplifies the analysis and make it possible to derive a clean model for the transition probability and the bounding (or stationary) distribution of the random walk of the graph, it is natural to consider different random walks with mixed values for the same parameter – assigned locally by each node depending on its perception of the overall graph and trust in it. Providing clean formulation for this node-wise (as opposed to graph-wise) parameters stay an open question that we would like to investigate further. This particularly will be more meaningful in designs that do not use random walks for their operation, and would be more challenging in designs that operate on other measures, such as centralities. Indeed, investigating how a node-wise parameter, as opposed to network-wise parameter is an area worth of investigation in its own right.

- We want to extend these results and findings by applying these designs to other Sybil defenses, other routing algorithms—our initial results not shown this thesis which tested how a simple variation of these designs impacts shortest path and random walk routing, as well as gossiping, is in [**?**].

Finally, we use social networks for building two types of applications that make use of new properties discovered in these networks. The first application is a distributed computing service (called SocialCloud) that does not make any direct use of a global property of social graphs (such as the mixing time or the expansion). We show that nodes in SocialCloudfinish computing tasks relatively quickly even when as much as 30% of the nodes in the system are outsourcing computations. The second system we design is DynaMix, an anonymous communication service that incorporates dynamics in the social graph for anonymity. We show that dynamics of these networks can be used to improve anonymity of users. This direction enables several future works that go beyond this thesis, including the following:

- SocialCloudoes not include several scenarios that are natural in distributed computing services, including, for example, failure. We aim to complete the missing ingredient of the simulator and enrich it by further scenarios of deployment of our design, under failure, with different scheduling algorithms at both sides of the outsourcer and workers (in addition to those discussed in the work), and to consider other overhead characteristics that might not be in line with topological characteristics in the social graph. These characteristics may include the uptime, downtime, communication overhead, and I/O overhead consumption, among others. One interesting feature that we will consider is trust-based scheduling, benefiting from the prior work in [**?**].

- We will turn our attention from the simulation settings to real-world deployment settings, thus addressing options discussed in §6.5.7, and to implement a proof-of-concept application, among those discussed in §6.2.1, by utilizing design options discussed in this work. We anticipate a lot of hidden complexities in the design to arise, and significant findings to come out of the deployment that we will report on in the future work.

- Dynamix, in its entirety, has considered "unstructured" social graphs, which a non-constant (and likely power-law) degree distribution. On one hand, the potential of structured graphs for anonymity is well studied, and beautiful theoretical results are already provided. On the other hand, suitability of these results for real-world social structures, especially when considering dynamics, is unclear. In the future, we will look at creating structured graphs from unstructured social graphs, and explore their potential for anonymity systems.